Spring 2.0 技术手册

林信良 著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

内 容 简 介

本书介绍了 Spring 2.0 的新特性，诸如 Spring 2.0 的新配置、新 AOP 支持、增强的 IoC、JDBC 和 form 标签等新功能。它通过实际完成一个完整的 Spring 项目示例，展示了与 Spring 相关 API 的使用技巧，能够显著减少每一位入门者摸索 Spring API 的时间。本书是学习 Spring Web 开发的最佳读物，它能让读者在示例学习中获得显著提高。

本书内容全面深入，主要包括 Spring 入门、Bean/消息/事件、Spring AOP、JDBC/事物支持、Spring 与 Hibernate 的整合、Spring Web MVC 框架、View 层方案/Web 框架整合、API 封装、Spring 在线书签完整项目示例等内容。

本书为台湾碁峰资讯股份有限公司独家授权发行的中文简体版。本书中文简体字版在中国大陆之专有出版权属电子工业出版社所有。在没有得到本书原版出版者和本书出版者书面许可时，任何单位和个人不得擅自摘抄、复制本书的一部分或全部以任何方式（包括资料和出版物）进行传播。本书原版版权属碁峰资讯股份有限公司。版权所有，侵权必究。

版权贸易合同登记号　图字：01-2007-1376

图书在版编目（CIP）数据

Spring 2.0 技术手册 / 林信良著.—北京：电子工业出版社，2007.4
ISBN 978-7-121-03985-0

I. S… Ⅱ.林… Ⅲ.JAVA 语言－程序设计－技术手册 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2007）第 031880 号

责任编辑：周　筠　杨绣国
印　　刷：北京市铁成印刷厂
装　　订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：28.75　字数：550 千字
印　　次：2007 年 4 月第 1 次印刷
定　　价：49.80 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系电话：（010）68279077；邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

读者对《Spring 技术手册[1]》的赞誉

“我很喜欢第 2、3、6、8 这几章。在看此书之前我看了一些 Spring 的资料，可有些东西还是不很清楚。看过本书后有种豁然开朗的感觉，原来不懂的一些东西一下子就全明白了，感觉很不错。”

——沈阳市读者

“最喜欢第 3、4、5、6、9、10 章，以及第 8 章的部分内容。此书易懂，真的很棒，个人觉得《Spring 技术手册》是市面上最易懂的 Spring 书籍（注意，没有之一）。”

——大连市读者

“最喜欢‘Bean、消息、事件’这一章，内容充实，实例鲜明！”

——哈尔滨市读者

“很喜欢第 6、8 章，因为这两章涉及 Spring 和 Hibernate、JSF 相结合的内容，对现实应用有很好的帮助。”

——北京市读者

“其实，这本书的每章都非常棒。特别是里面的示例，很简单也很容易实现，很棒！”

——南京市读者

“我最喜欢第 7 章中讲的 Spring Web MVC 框架，它让我弄清楚了从请求到请求分派到 Controller 实现类的过程，谢谢！”

——广州市读者

“我最喜欢第 4 章的 AOP，以及第 7 章 Spring 的 MVC 内容。”

——汕头市读者

[1] 本书是对畅销书《Spring技术手册》的全新改版，涵盖Spring 2.0的新特性。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

“林信良的作品肯定不错，非常适合初学者，刚入门 Spring 的程序员非常适合看这本书！这本书带我进入了 Spring 的世界。”

——www.dearbook.com 读者

我看了 4 本已出版的与 Spring 相关的书籍：《Spring 高级框架设计》（自购）、《Spring in Action》（借阅）、《Spring 技术手册》（自购）、《精通 Spring》。

对于像我这样的 Spring 初学者（懂点设计模式，也有一定的代码积累和一点项目经验）来说，最理想的书可能就是《Spring 技术手册》。可不要被书名迷惑，这可决不是一本册子这么简单，它对 Ioc、AOP 的分析是这 4 本书中最容易懂的了（作者在讲解之前做了一些技术和概念的铺垫），如果你是初学者，强烈推荐。

——www.china-pub.com 读者

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

夏昕、林信良谈 Spring 框架

CSDN、博文视点名家讲坛邀请夏昕先生和林信良先生针对 Spring 框架进行了在线访谈。现将谈话内容整理成文，以飨读者。

采 访 人：《程序员》杂志的技术编辑霍泰稳

受 访 人：夏昕、林信良

采访方式：CSDN 聊天室在线访谈 *http://chat.csdn.net*

♠ 主持人：

大家好，随着夏昕老师的译作《Spring 专业开发指南》、林信良老师的《Spring 2.0 技术手册》的上市，Spring 继续成为大家关注的热点。今天我们有幸请到两位老师来参加 CSDN F2F 俱乐部和博文视点资讯有限公司共同举办的名家讲坛活动，本次活动会就“轻量级框架——Spring”这一话题与大家在线交流。

这里先简单介绍一下两位专家的背景：

夏昕是业界知名的技术专家、金融行业的技术顾问、OpenDoc 项目发起人及《DrBobb's Journal China》的常任编委，著有广受好评的畅销书《深入浅出 Hibernate》。

林信良老师（网名为良葛格）曾是台湾 SUN 教育训练中心讲师，著有《Spring 2.0 技术手册》、《JAVA 学习笔记》。他是 SUN 官方教材的主要编写成员之一。林老师的博客地址是：*http://blog.csdn.net/ caterpillar_here/*。

下面请夏昕老师、林信良老师开始回答网友们的提问。

♠ 主持人：

第一个问题，新手到达什么程度可以开始 Spring 的学习？

♣ 夏昕：

一般来说，如果拥有一年多持续的 Java Web 应用开发经验，学习 Spring 基本上就没太大问题。目前公司团队中，几乎所有成员都是从进入公司开始学习应用 Spring 的。从程度来看，一年的实际 Web 开发经验已足以打下理解 Spring 基础结构的基础。这时候，只要有适当的资料加以辅助，学习起来就不会太难了，呵呵。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

♠ 主持人：

网友提问，Spring 能够粘合不同层面的解决方案，但无论是 MVC 层还是 ORM 层，Java 领域从来都不缺少好的框架，我们在使用 Spring 时，在这些框架中作选择的依据是什么？

♥ 林信良：

选择框架时，主要是考虑稳定性。例如 Struts，虽然它是个相对古老的框架，但稳定性高。也许就现在的设计观点来说它旧了，但开发不是追求流行，不是说最新的框架就是最好的。另外，选择框架的一个重点常被忽略，那就是人员培训。一个好但复杂的框架或一个好但缺乏文档的框架，能发挥它的功用吗？开发不是一两个人的事，而是众人的事。我看过很多公司为了追求流行而忽略了这一点。

♠ 主持人：

网友提问，与仅仅使用 JSP 或 Struts 等简单框架进行开发相比，使用 Spring 的 Web 应用性能如何？

♣ 夏昕：

客观来讲，Spring 的微内核容器的确会在一定程度上消耗额外的系统性能。不过，除非是开发具有并发性能（比如超大规模论坛）、实时性能（比如超大规模信用卡在线交易系统）等关键业务系统，否则我还是推荐使用 Spring。Spring 为我们带来的系统结构和开发上的帮助，远远可以抵消这部分损失。

♠ 主持人：

网友提问，夏老师是金融行业的顾问，想问一下，您是怎么说服金融行业的人员使用开源产品的，我所遇到的一些金融行业从业人员对开源甚至 Java 都不愿意接受，他们更喜欢古老但坚固的 C++。

♣ 夏昕：

这可能是传统思路的延续吧，跟国内金融系统一贯的风险规避思路有关。我为花旗银行亚太数据中心工作的时候，看到很多特关键的业务，其后台都是 Tomcat 在运行。所以，开源产品本身并没有什么问题，关键还在于思路的改变，这需要时间。

♠ 主持人：

网友提问，学习 J2EE 开发的人，不学习 Spring 可以么？

♥ 林信良：

可以啊！并没有规定一定要学过 J2EE 才可以学习 Spring，也没有规定一定要学过 Spring 才可以学习 J2EE。在学习 Spring 甚至是一些 opensource project 时，重点是了解其精神。学习 Spring 有助于理解 J2EE 开发中的设计，事实上，opensource 界在 Java 的设计精神上取得了很大的胜利，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在未来的 Java EE 中，有很多都会借鉴 opensource 界的经验，比如采用 Spring/Hibernate 的一些设计精神。在这种情况下，学习 Spring，或者学过 Spring/Hibernate 等，多少都会有助于了解将来的 Java EE。

♠ 主持人：

网友提问，什么情况下适合将 Spring 纳入项目中加以应用（最适合 Spring 应用的场景有哪些）？

♥ 林信良：

使用 Spring，最怕的是为了 Spring 而 Spring，为了 IoC 而 IoC，为了 AOP 而 AOP，这样的话，Spring 对您没有好处。有几点可以参考：

1. 当组件出现重用时，再开始考虑使用 Spring IoC。
2. 当组件用到外部资源时，使用 Spring IoC。
3. 当组件需要 AOP 时，再考虑设置 Spring xml。
4. 能不用 Spring IoC 就不用！
5. 不要为 Spring 而 Spring。
6. 不要为 IoC 而 IoC。
7. 不要为 AOP 而 AOP。

简单地说，采用 XP 的方式可以让您考虑何时该采用 Spring，何时该采用 IoC、AOP 等。这里有个案例可以参考：*http://www.javaworld.com.tw/roller/page/ingramchen?entry=2006_2_1_KeepSpringIoCCoarseGrained*。另外，Spring 在 JDBC 这方面所提供的封装也值得分开来看看，可以省去不少功夫。

♠ 主持人：

网友提问，Spring 里面的接口编程有哪些好处？如果用类来实现 Service，DAO 层同样可以实现注入吗?为什么要接口呢?有的资料说可以重用代码，但是对于一般的 Service，DAO 的接口就只被一个类实现，何谈重用呢?

♣ 夏昕：

这就跟电脑上的 USB 接口一样，一般这个接口只会插鼠标，如果什么时候需要用 U 盘了，插上也可以。Spring 提供了这种实现的基础，至于重用性本身是否存在，那可能就须要在开发项目前进行调研了。比如做一个只用一次的投票表单，用一个 JSP 搞定更干净利落，这时重用性并不重要。

♠ 主持人：

网友提问，其公司的业务一般都使用最简单的 JSP+Beand 技术。这样，接触 Struts 和 Spring 的机会就很少了，可否请林老师提些建议，如何在这种环境下学好 Spring。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

♥ 林信良：

公司如果不使用 Spring，并不是什么坏事，我看过因为用了 Spring/Hibernate 而使得学习成本大幅提高的情况。如果你们公司业务完全不用 Spring 或 Sturts 等技术，那您就需要个人学习 Spring 了，没必要为了用这些东西在公司大力倡导。您可以自学，将来有机会自然就会用上，到时您所学的知识就可派上用场了。

♠ 主持人：

网友提问，请问夏老师对 EJB 3.0 怎么看，EJB 3.0 从开源框架里汲取了不少灵感，甚至有一些开源大师也参与其中，那 EJB 3.0 会和 Spring、Hibernate 形成竞争关系吗？

♣ 夏昕：

EJB 3.0 实际上包括了 Hibernate，所以它们不存在设计思想上的竞争。至于 Spring，竞争可能会比较激烈一点，呵呵。

♠ 主持人：

网友提问，Spring 框架是否是更适用于 B/S 项目？

♥ 林信良：

Spring 框架提供各种子框架，并不局限于 B/S 项目，您也不用将 Spring 中的子框架全部用上，您可以根据项目需要，采用其中适当的子框架。不必为了用 Java 而把 Java SE 的所有 API 全部都用上，不是吗？

♠ 主持人：

网友提问，是否可以通俗地解释一下“依赖注入”，谁依赖谁了？

♥ 林信良：

依赖注入指的是，当您在 A 对象内部需要 B 对象的某个功能时，A 就依赖于 B，您可以直接在 A 内部实例化 B，但日后要修改就麻烦些。如果您不直接在 A 内部实例化 B，而是通过 IoC 将一个 B 实例从外部设置给 A，那就比较有弹性。举个通俗例子，如果将 CPU 固定在主板上，那主板就依赖于 CPU，若保留 CPU 的插槽规格，让 CPU 可以自由更换，就有点 IoC 的味道了。

♠ 主持人：

网友提问，Spring 的 MVC framework 不算是整个 Spring 的核心部件，但它的威力却超出了很多人的想象。很多人认为 Spring 的 MVC framework 是非常优秀的，甚至比 Webwork 2 更优秀。两位老师是如何评价的呢？

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

♥ 林信良：

就我个人而言，使用 Spring MVC 最大的好处是，可以直接整合并善用其 IoC 容器功能，Spring MVC 与 Struts 的设计其实是蛮像的，至于其他部分，我并不觉得 Spring MVC 有什么特别突出之处。

♣ 夏昕：

呃，我觉得 Spring MVC 是以干练取胜，它能以极其简炼的办法实现 Control 逻辑。喜欢简洁的人会非常喜欢 Spring MVC 的实现方式。但是它在某些方面还是存在一些缺陷，比如 wizard 的 backspace 问题一直没能很好地解决。另外，在实际应用过程中跟 Spring Webflow 的配合也总是比较尴尬。我们公司目前选择的是 Webwork+Spring+Hibernate 的组合。

♠ 主持人：

网友提问，Spring 和轻量级、轻量级和重量级的概念是如何划分的？Spring 对 JTA、EJBSB、JMS、WebService 都支持，而且也都封装得很好。

♣ 夏昕：

现在轻量级和重量级的确很难区分了，标准越来越多，也越来越难以衡量。之前有人说，有 classloader 的算是重量级；也有人说，超过 3MB 的算是重量级，真的很难说了。EJB 有轻量化的倾向，Spring 也有重量化的趋势。简单说吧，Spring core 算是轻量级的，但是加上其余的部件，估计就得列入重量级的范畴了，呵呵。

♠ 主持人：

网友提问，怎么设计系统架构才能成功实现轻量级开发？是不是用了 Spring 就是轻量级开发了呢？

♥ 林信良：

我觉得这个问题提得非常好！常有人问，这个功能用 Spring 怎么实现？那个功能用 Spring 如何实现？事实上，我心里想的就是，要实现某个功能，自己写个组件会比直接用 Spring 来得好！用了 Spring 并不等于轻量级开发。重点是您有没有采纳其精神，用了 IoC 容器就轻量级了吗？不见得。什么都来个 DI，我看也是很沉重的。如果您在设计时善用 IoC，即使不用 Spring 也是轻量级。

♠ 主持人：

网友提问，现在 Spring 添加了许多的功能，这样下去岂不是也变得不轻量了么？

♥ 林信良：

您什么都要用 Spring 来做，那就是重量级了。还是那句话，不要为了 Spring 而 Spring！如果 Spring 开始让您觉得沉重，那它就不是个适用的框架，谨记！如果在采用 Spring 前，您的 team 就开始叫苦连天，而且还要先花一堆时间去学，但又无法入门，那就不要用。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

♠ 主持人：

网友提问，Spring 和 Struts 在运用方面各有什么特点？

♥ 林信良：

Spring 与 Struts 基本上是两个目的不同的框架 您要比的应该是 Spring MVC 与 Struts 的特点！Struts 的特点是成熟，它已经是 Model 2 的代表了，它的可用组件多、Tag 丰富。而 Spring 的 View 层可以适当地抽换，使用 JSTL 可以整合其 IoC 功能等，这就是其特点。

♠ 主持人：

网友提问，如果现在要开发一个新的 B/S 项目，两位推荐的 MVC 组合是什么？（？+ Spring + ？）

♥ 林信良：

您指的已经不只有 MVC，而是层与层的设计了，Web 层现在有很多选择了，Struts/Webwork 2 都是不错的，至于 Persistence 层，现在主流是 Hibernate，而学习 Hibernate，有助于了解未来的 EJB 3.0，这点或许可以列入考虑中。

♠ 主持人：

网友提问，请问夏老师☺，怎样修炼才能成为系统架构师？

♣ 夏昕：

这里就不说具体的方面了。我觉得，首先是要有好奇心和钻研精神。大家今天能来这里，相信都不缺乏这些，另外很重要的一点就是总结和分析的能力。之前我发布的那些 open .doc，都是在研究某个框架时随手记录的一些经验总结，类似每天一篇 blog 的形式，最后汇总起来，也可以成为一本书籍。《深入浅出 Hibernate》就是这么诞生的。有什么收获，写下来，与大家分享，可以使自己进步更快。这一点是我最大的体会。

♠ 主持人：

网友提问，请问夏昕老师是哪里人呀？请您说说您的 Java 学习之路。

♣ 夏昕：

呵呵，扬州人。在学习 Java 之前，做过几年 VC++开发，后来转到 Java 也是很奇怪的事情。好像是 Java 项目组里的人不够了，刚好这时候自己的事已做完，于是也就凑个热闹写了两个 Java Socket 程序。后来 Java 项目组里的人总是不够，于是……呵呵。基本算是在实战中学习的吧。

♠ 主持人：

非常感谢两位老师的精彩回答与奉献精神，敬礼！！！

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

序

对我来说，写书是一种想法的实现过程。书稿与我在网络上所放置的笔记文件不同，笔记文件只是学习过程的一个记录，而在写作成书时，需要考虑以什么方式呈现书的内容、呈现哪些内容，以及传达什么样的信息。

在写作《Spring 技术手册》时，我的想法是总结从实践中学习的成果，当 Spring 2.0 接近发布之时，我又在思考：驱动《Spring 技术手册》改版的想法是什么。

《Spring 技术手册》问世之后，收到了很多建议与批评，也有幸受邀参与了 Spring 相关课程的交流讨论。从这些回馈与交流当中，我逐步明确了将《Spring 技术手册》改版为《Spring 2.0 技术手册》的思路与轮廓。

最开始的想法是在改版时，要将 Spring 2.0 的所有特性详加说明，然而一来个人能力有限，二来 Spring 的参考文档中已将自己想到的很多东西论述得很清楚了，若再要置喙，只是沦于抄写翻译之工而已。所以最后决定重点介绍 Spring 2.0 的新特性。另一个想法是，在几次 Spring 相关课程的交流中，我有了一些教学经验、观念与实践方式的积累，想将这些东西加入书中。

《Spring 2.0 技术手册》仍然定位于初学者，它是引导初学者入门的敲门砖，希望初学者通过这块敲门砖，得以进入 Spring 的大门，进而探索门后的宝库。《Spring 2.0 技术手册》之目的，就是让您在阅读完之后，能够自行查阅 Spring 参考文档，看懂更进阶的书籍，以及参与网络上关于 Spring 2.0 的深入讨论，真正踏上 Java 开发之路。

林信良

2006 年 11 月 9 日

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

目录

导　读 I

第 1 章　认识 Spring 1

1.1　术语介绍 2

1.2　控制反转（Inversion of Control） 4

1.3　依赖注入（Dependency Injection） 9

1.4　接下来的主题 11

第 2 章　Spring 入门 13

2.1　第一个 Spring 程序 14

2.1.1　下载、设置 Spring 14

2.1.2　准备 Spring 设计环境（使用 Eclipse） 16

2.1.3　编写第一个 Spring 程序 20

2.1.4　编写第二个 Spring 程序 23

2.1.5　Spring 2.0 的 XML 设置 26

2.2　安装、使用 Spring IDE 27

2.2.1　安装 Spring IDE 27

2.2.2　使用 Spring IDE 29

2.3　接下来的主题 32

第 3 章　Bean、消息、事件 35

3.1　Bean 基本管理 36

3.1.1　BeanFactory、ApplicationContext 36

3.1.2　Bean 的识别名称与别名 39

3.1.3　Bean 的实例化 40

3.1.4　Bean 的 scope 42

3.1.5　Bean 的生命周期 43

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.1.6 Bean 定义的继承 45
3.2 Bean 的依赖设置 48
3.2.1 Type 2 IoC、Type 3 IoC 48
3.2.2 依赖的值设置与参考 52
3.2.3 自动绑定 55
3.2.4 集合对象 60
3.2.5 Spring 2.0 的<util>标签 68
3.2.6 Lookup Method Injection 71
3.3 Bean 高级管理 73
3.3.1 非 XML 定义文件的配置方式 74
3.3.2 Aware 相关接口 76
3.3.3 BeanPostProcessor 78
3.3.4 BeanFactoryPostProcessor 81
3.3.5 PropertyPlaceholderConfigurer 81
3.3.6 PropertyOverrideConfigurer 83
3.3.7 CustomEditorConfigurer 84
3.4 资源、消息、事件 88
3.4.1 资源的取得 88
3.4.2 解析文字消息 89
3.4.3 监听事件 92
3.4.4 事件传播 93
3.5 接下来的主题 95
第 4 章 Spring AOP 97
4.1 AOP 入门 98
4.1.1 从代理机制初探 AOP 98
4.1.2 AOP 观念与术语 105
4.1.3 Spring AOP 108
4.2 Advices 109
4.2.1 Before Advice 109
4.2.2 After Advice 113
4.2.3 Around Advice 115
4.2.4 Throw Advice 117
4.3 Pointcut、Advisor 121

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.3.1 NameMatchMethodPointcutAdvisor 121
4.3.2 RegExpMethodPointcutAdvisor 124
4.3.3 ControlFlowPointcut 126
4.3.4 Pointcut 接口 129
4.3.5 Pointcut 交集、并集操作 131
4.4 Introduction 132
4.4.1 IntroductionInterceptor 133
4.4.2 DelegatingIntroductionInterceptor 137
4.5 Autoproxing 141
4.5.1 BeanNameAutoProxyCreator 141
4.5.2 DefaultAdvisorAutoProxyCreator 142
4.6 Spring 2.0 的 AOP 支持 144
4.6.1 Before Advice：基于 XML Schema 144
4.6.2 Before Advice：基于 Annotation 147
4.6.3 Spring 2.0 的 Pointcut 定义 149
4.6.4 After Returning Advice：基于 XML Schema 151
4.6.5 After Returning Advice：基于 Annotation 153
4.6.6 After Throwing Advice：基于 XML Schema 155
4.6.7 After Throwing Advice：基于 Annotation 158
4.6.8 Around Advice：基于 XML Schema 159
4.6.9 Around Advice：基于 Annotation 161
4.7 接下来的主题 162
第 5 章 JDBC、事务支持 163
5.1 Spring 持久层入门 164
5.1.1 Spring 的 DAO 支持 164
5.1.2 DataSource 注入 169
5.1.3 DataSource 置换 175
5.2 JDBC 支持 177
5.2.1 使用 JdbcTemplate 177
5.2.2 执行与更新 JdbcTemplate 179
5.2.3 JdbcTemplate 查询 183
5.2.4 JdbcTemplate 的 Lob 支持 185
5.2.5 以对象方式进行操作 187

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.2.6 DataFieldMaxValueIncrementer 192
5.2.7 Spring 2.0 的 NamedParameterJdbcTemplate 192
5.2.8 Spring 2.0 的 SimpleJdbcTemplate 194
5.3 JDBC 事务管理 196
5.3.1 Spring 对事务的支持 197
5.3.2 JDBC 编程事务管理 199
5.3.3 JDBC 声明事务管理 202
5.3.4 事务的属性介绍 206
5.3.5 TransactionAttributeSource、TransactionAttribute 208
5.3.6 Spring 2.0 声明式事务管理：基于 XML Schmea 212
5.3.7 Spring 2.0 声明式事务管理：基于 Annotation 214
5.4 接下来的主题 216
第 6 章 Hibernate 与 Spring 219
6.1 Hibernate 入门 220
6.1.1 简介 Hibernate 220
6.1.2 下载、设置 Hibernate 222
6.1.3 第一个 Hibernate 程序 224
6.2 在 Spring 中整合 Hibernate 231
6.2.1 SessionFactory 注入 232
6.2.2 HibernateTemplate 235
6.2.3 HibernateTemplate 的 Lob 支持 237
6.2.4 Hibernate 编程事务管理 240
6.2.5 Hibernate 声明事务管理 242
6.3 接下来的主题 248
第 7 章 Spring Web MVC 框架 249
7.1 Spring Web MVC 入门 250
7.1.1 使用 Eclipse Web Tools Platform 250
7.1.2 第一个 Spring Web MVC 程序 253
7.1.3 WebApplicationContext 257
7.1.4 Handler Mapping 261
7.1.5 Handler Interceptor 263
7.1.6 Controller 继承架构 266

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.1.7 ModelAndView 269
7.1.8 View Resolver 271
7.1.9 Exception Resolver 274
7.2 Controller 实现类 275
7.2.1 AbstractController 275
7.2.2 MultiActionController 与 ParameterMethodNameResolver 277
7.2.3 MultiActionController 与 PropertiesMethodNameResolver 280
7.2.4 ParameterizableViewController 284
7.2.5 AbstractCommandController 285
7.2.6 AbstractFormController 288
7.2.7 SimpleFormController 289
7.2.8 AbstractWizardFormController 294
7.2.9 ThrowawayController 300
7.3 搭配 Controller 的相关类 301
7.3.1 实现 Validator 301
7.3.2 使用 PropertyEditor 304
7.3.3 文件上传 309
7.4 接下来的主题 313
第 8 章 View 层方案、Web 框架整合 315
8.1 结合 JSTL 与 Spring 标签 316
8.1.1 结合 JSTL 316
8.1.2 <spring:bind> 标签 319
8.1.3 数据绑定的几个方法 322
8.1.4 <spring:message> 标签 325
8.1.5 <spring:transform> 标签 327
8.1.6 Spring 2.0 的 form 标签 332
8.2 其他 View 技术 334
8.2.1 以 Tiles 为例 334
8.2.2 自制 View Class 339
8.3 整合 Struts 框架 342
8.3.1 第一个 Struts 程序 342
8.3.2 在 Struts 中整合 Spring 346
8.4 整合 JSF 框架 351

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.4.1 第一个 JSF 程序 351
8.4.2 在 JSF 中整合 Spring 358
8.5 接下来的主题 362
第 9 章 API 封装 363
9.1 远程 364
9.1.1 RMI 364
9.1.2 Hessian、Burlap 368
9.1.3 Http Invoker 371
9.2 邮件 373
9.2.1 简单邮件 373
9.2.2 HTML 邮件 374
9.2.3 内嵌图片或附件 376
9.3 任务计划 379
9.3.1 使用 TimerTask 379
9.3.2 使用 MethodInvokingTimerTaskFactoryBean 381
9.3.3 使用 Quartz 383
9.3.4 使用 MethodInvokingJobDetailFactoryBean 387
9.4 接下来的主题 389
第 10 章 项目：Spring 在线书签 391
10.1 程序概观 392
10.1.1 架构规划 392
10.1.2 采用技术 393
10.1.3 功能概述 394
10.2 Model 设计 394
10.2.1 UserDAO 394
10.2.2 BookmarkDAO 398
10.2.3 SimpleMail 401
10.3 View 设计 402
10.3.1 Tag Files 402
10.3.2 注册网页 404
10.3.3 登录网页 407
10.3.4 会员网页 409

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

10.3.5 加入书签网页 410
10.3.6 改变、取回密码网页 411
10.3.7 首页重新导向 413
10.4 Controller 设计 414
10.4.1 RegisterController 414
10.4.2 LoginController 418
10.4.3 MemberController 420
10.4.4 ForgotController 424
10.5 配置设置 426
10.5.1 设置 web.xml 426
10.5.2 设置 web-config.xml 427
10.5.3 设置 model-config.xml 429
10.6 接下来的主题 430

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

导读

这份导读可以让您更了解如何使用本书。

本书许多的范例都通过完整的程序实现来展示，在以下的程序代码范例中：

SpringDemo　　HelloBean.java

```
package onlyfun.caterpillar;

public class HelloBean {
    private String helloWord;

    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }

    public String getHelloWord() {
        return helloWord;
    }
}
```

范例左上角的名称“SpringDemo”表示可以在光盘中找到对应的 SpringDemo 项目，而右上角的名称“*HelloBean.java*”表示可以在项目中找到 *HelloBean.java* 文件。

如果按以下方式呈现代码，表示它是一个完整的程序，但不属于范例项目的一部分，主要用来展示如何编写一个完整的文档：

```
helloBean.class=onlyfun.caterpillar.HelloBean
helloBean.helloWord=Welcome
```

如果按以下方式呈现代码，则表示它是个程序代码片段，在编写程序时需要特别注意：

```
...
<beans...>
    <bean id="beanFactoryModifier"
          class="onlyfun.caterpillar.SomeClass"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    ...
</beans>
...
```

由于书籍的页面宽度有限，程序代码有时会因为过长而无法容纳于书籍的一行之中，此时不得不隔行表示，这时会使用“➥”符号，表示两行实际上是必须连接在一起的，例如：

```
...
    <bean id="configBean"
         class="org.springframework.beans.factory.
                ➥ config.PropertyPlaceholderConfigurer">
        <property name="location">
            <value>hello.properties</value>
        </property>
    </bean>
...
```

在上面的程序片段中，"class"属性的完整程序代码编写实际上应该是org.springframework.beans.factory.config.PropertyPlaceholderConfigurer。

在本书中会出现以下的对话框：

良葛格的话匣子<<<

在这个对话框中提供的可能是良葛格的一些主观性较强的意见、经验或建议，在这个对话框中也有可能提供一些主题外的常识，或一些额外的参考文件。

本书提供符合 Eclipse 项目结构的源文件，而对于 Web 应用程序的范例，则提供符合 Eclipse Web Tools Platform（WTP）项目结构的源文件。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 1 章 认识 Spring

Spring 的核心是个轻量级（Lightweight）的容器（Container），它是实现 IoC（Inversion of Control）容器和非侵入性（No intrusive）的框架，并提供 AOP（Aspect-oriented programming）概念的实现方式；提供对持久层（Persistence）、事务（Transaction）的支持；提供 MVC Web 框架的实现，并对于一些常用的企业服务 API（Application Interface）提供一致的模型封装，是一个全方位的应用程序框架（Application framework），除此之外，对于现存的各种框架（Struts、JSF、Hibernate 等），Spring 也提供了与它们相整合的方案。

对于打算入门 Spring 的初学者来说，面对轻量级、非侵入性、容器、IoC、AOP 等术语往往不知所措。在这个章节中，将针对这些术语作一些介绍，作为往后各个章节中介绍 Spring 对这些术语概念实现时的基础。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

1.1 术语介绍

Spring 的目标之一，是作为一个全方位的应用程序框架（Application framework），由于目前应用程序在各个领域中的应用越来越复杂，因而各个领域在设计与实践上的概念与术语也越来越多。作为一个全方位的应用程序框架（Framework），在使用 Spring 时就得接触到像是轻量级、容器、非侵入性（No intrusive）、IoC（Inversion of Control）、AOP（Aspect-oriented programming）等术语，对于初学 Spring 的读者而言，以下的说明可以让您对这些术语与概念有个基本的认识。

- 轻量级（Lightweight）

 轻量级是相对于一些重量级的容器（如 EJB 容器）来说的，Spring 的核心包在文件容量上只有不到 1MB 的大小，使用 Spring 核心包所需要的资源也是很少的，您甚至可以在小型设备中使用 Spring。

- 非侵入性（No intrusive）

 框架原来的用意是提供一个架构的实现，让开发人员可以在基于框架的基础上，快速地开发出遵循架构所需的应用程序，然而有些框架一旦被使用，应用程序就对框架有了依赖性，例如使用了大量框架的 API，或直接继承 API 的某些类型等，都会使应用程序组件依赖于框架，而无法从框架中独立出来，更别说当中的组件可以直接重用于另一个应用程序之中了。

 Spring 的目标之一是实现一个非侵入性（No intrusive）的框架，希望让应用程序几乎感受不到框架的存在，减少应用程序从框架移植时的负担，进一步增加应用程序组件的可重用性（Reusability）。简单地说，如果使用 Spring，应用程序中的某些组件可以直接拿到另一个应用程序或框架之中使用。

- 容器（Container）

 容器可以管理对象的生成、资源取得、销毁等生命周期，甚至建立对象与对象之间的依赖关系。Spring 提供容器功能，您可以使用一个配置文件（通常是 XML），在当中确定对象名称、确定如何产生对象（Prototype 方式或 Singleton 方式）、哪个对象产生之后必须设定成为某个对象的属性等。在启动容器之后，所有的对象都可以直接取用，不用编写任何一行程序代码来产生对象，或是建立对象与对象之间的依赖关系。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

具体从 Java 的角度来说，容器就是一个 Java 所编写的程序，原先必须自行编写程序以管理对象关系，现在容器都会自动帮您做好。

- Inversion of Control 与 Dependency Injection

Spring 最重要的核心概念是 Inversion of Control，中文常译为“控制反转”，简称 IoC。在 Spring 中，“依赖关系的转移”、“依赖于抽象而非实践”是重要的观念，从对象的角度来说，可以避免对象之间的耦合；从容器的角度来说，可以避免应用程序因依赖于容器的功能，而从容器脱离。

Spring 另一个重要核心概念为 Dependency Injection，中文常译为“依赖注入”，简称 DI。使用 Spring，不必自己在程序代码中维护对象的依赖关系，只需在配置文件中加以设置，Spring 核心容器会自动根据配置，将依赖注入指定的对象。

在下一节中，还会详细介绍 Inversion of Control 与 Dependency Injection 的概念。

- AOP（Aspect-oriented programming）

Spring 最被人重视的功能之一是支持 AOP（Aspect-oriented programming）的实现，然而 AOP 框架只是 Spring 支持的一个子框架，说 Spring 框架是 AOP 框架并不是一个适当的描述，人们对于 AOP 的关注反映在 Spring 上，这使得人们对于 Spring 的关注集中在它的 AOP 框架上，虽然有所误解，但也突显了 Spring 的另一个令人关注的特色。

举个实际的例子来说明 AOP 的应用，假设应用程序有个日志（Logging）的需求，您可以无须修改任何一行程序代码，就将这个需求加入至原先的应用程序中，而若您愿意，也可以在不修改任何程序的情况下，将这个日志功能移除。

Spring 的 IoC 容器功能与 AOP 功能的实现是其重心所在，在 Spring 中实现了持久层、MVC Web 框架以及各种企业服务的 API 封装，它们的实现有些依重于 Spring 的 IoC 容器与 AOP 功能，Spring 的这些子框架或封装的 API 功能彼此可以独立，也可以结合其他的框架方案加以替代，Spring 希望提供 one-stop shop 的框架整合方案。

- 持久层

Spring 提供对持久层的整合，如对 JDBC 的使用加以封装与简化，提供程式事务（Programmatic Transaction）与声明式事务（Declarative Transaction）管理功能。对于 O/R Mapping 工具（Hibernate、iBATIS）的整合及使用上的简化，Spring 也

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

提供了对应的解决方案。

- Web 框架

 Spring 也提供 Model 2（Web MVC）框架的解决方案，使用 Spring Web 框架的好处是可以利用其本身 IoC 与 AOP 的功能，轻松地替换使用不同的 View 层技术，例如使用 JSP、结合 Tiles、使用 PDF 作为展现给用户的显示技术。

 您也可以将自己所熟悉的 Web 框架与 Spring 整合，像 Struts、JSF 等，都可以与 Spring 整合，来适用于当前所进行的应用程序。

- 其他企业服务的封装

 对于一些服务，例如 JNDI、Mail、任务计划（Scheduling）、远程（Remoting）等，Spring 不直接提供实现，而是采取抽象层方式对这些服务进行封装，让这些服务有一致的使用模型，并且在使用上更为简化。

良葛格的话匣子<<<

如果您没有使用其他框架的经验，只是希望通过 Spring 的学习来吸收前人开发的经验，面对以上这些术语可能还是不太能了解，在 Spring 中这些概念都有具体的实现，您可以在往后的章节里从实践中体会。

1.2 控制反转（Inversion of Control）

Spring 的核心概念是 IoC，IoC 的抽象概念是“依赖关系的转移”。转移是相对于过去不良的应用程序设计来说的，像“高层模块不应该依赖于低层模块，而模块都必须依赖于抽象”是 IoC 的一种表现，“实现必须依赖抽象，而不是抽象依赖于实现”是 IoC 的另一种表现，“应用程序不应依赖于容器，而是容器服务于应用程序”也是 IoC 的一种表现。

一连串抽象的文字轰炸之后，到底 IoC 是什么意思？IoC 的全名是 Inversion of Control，如果硬要翻译成中文的话，就是“控制反转”。初看 IoC，从字面上不容易了解其意义，要了解 IoC，可先从 Dependency Inversion 开始了解，也就是依赖关系的反转，看看过去有哪些不良的依赖关系设计，应如何将这种不良的依赖关系反转，进而改进应用程序的可重用性。

简单地说，在进行模块设计时，高层的抽象模块通常是与业务逻辑（Business logic）相关的模块，它应该具有重用性，而不依赖于低层的实现模块。例如低层模块可能是与硬件相关的软盘存取设计，而高层模块是个存档备份的程序需求，如果高层模块直接调用执行低层模块的函式，就对低层模块产生了依赖关系。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

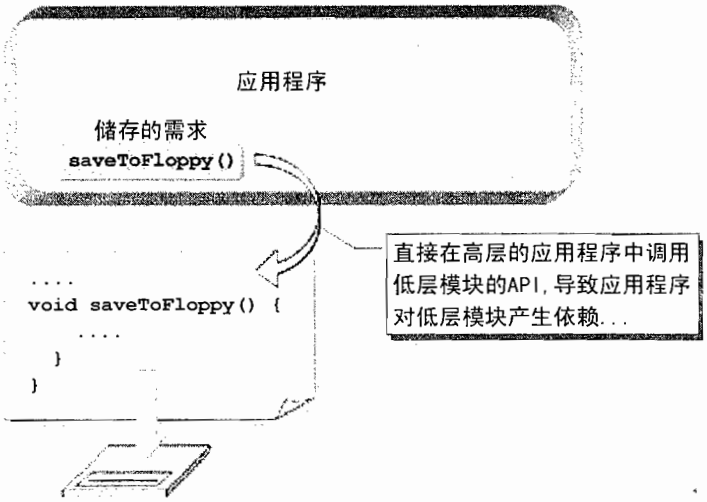


图 1.1 高层模块依赖低层模块实践

在图 1.1 中，应用程序在需要储存需求时直接执行了 saveToFloppy()，导致高层应用程序依赖于低层模块的 API。假设今天要将应用程序移植至另一个平台上，而该平台使用 USB 磁盘作为存储介质，则这个应用程序无法直接重用，必须加以修改才行。在这个例子中，由于低层模块存储介质的变化，造成了高层模块也必须跟着变化，这不是一个好的设计方式。在设计上希望模块都依赖于模块的抽象，这样才可以重用高层的应用程序设计。

若以对象导向的方式来设计，依赖反转（Dependency Inversion）解释为“程序不应依赖实现，而是依赖于抽象接口”。以 Java 程序为例：

```
public class Business {
    private FloppyWriter writer = new FloppyWriter();
    ....
    public void save() {
        ...
        writer.saveToFloppy();
    }
}
```

Business 类的设计中，存档的需求依赖于实际的 FloppyWriter 对象，如果今天想要将存储介质改为 USB 磁盘，则必须修改 Business 的程序，因为您无法直接重复使用 Business 类。

通过接口的声明，可以改进这一情况，例如可先声明一个 IDeviceWriter 接口：

```
public interface IDeviceWriter {
    public void saveToDevice();
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接着设计 Business 类。在遇到存档需求时，可以设计为依赖于 IDeviceWriter 接口，而不是依赖于实际的 FloppyWriter，例如：

```
public class Business {
    private IDeviceWriter writer;

    public void setDeviceWriter(IDeviceWriter writer) {
        this.writer = writer;
    }

    public void save() {
        ....
        writer.saveToDevice();
    }
}
```

在这样的设计下，Business 类就是可以重用的。如果今天有储存至 Floppy 或 USB 磁盘的需求，只要针对这两种储存需求分别实现 IDeviceWriter 接口即可，例如针对软盘的储存设计一个 FloppyWriter 类：

```
public class FloppyWriter implement IDeviceWriter {
    public void saveToDevice() {
        ....
        // 实际储存至 Floppy 的程序代码
    }
}
```

或是针对 USB 磁盘的储存设计一个 UsbDiskWriter 类型：

```
public class UsbDiskWriter implement IDeviceWriter {
    public void saveToDevice() {
        ....
        // 实际储存至 UsbDisk 的程序代码
    }
}
```

如果应用程序需要软盘储存，可以编写一个如下配置程序：

```
Business business = new Business();
business.setDeviceWriter(new FloppyWriter());
business.save();
```

同样的，如果应用程序需要 USB 磁盘存储，可以编写一个如下配置程序：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
Business business = new Business();
business.setDeviceWriter(new UsbDiskWriter());
business.save();
```

可以看到，无论低层的存储实现如何变动，对于 Business 类来说都无须作任何修改。您也可以编写一个配置管理程序，通过一个简单的 XML 或是 *.properties* 文件来更改配置，如此，连上面的配置程序都不必编写，就可以简单地更换 Business 类所依赖的 IDeviceWriter 实现，例如写个简单的 BusinessFactory：

```
package onlyfun.caterpillar;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class BusinessFactory {
    private static BusinessFactory factory;
    private Properties props;

    private Business business;
    private IDeviceWriter writer;

    private BusinessFactory() throws Exception {
        props = new Properties();
        props.load(new FileInputStream("config.properties"));
        String businessClass = props.getProperty("business.class");
        String writerClass = props.getProperty("writer.class");
        business = (Business) Class.forName(
                            businessClass).newInstance();
        writer = (IDeviceWriter) Class.forName(
                            writerClass).newInstance();
        business.setDeviceWriter(writer); // 建立依赖关系
    }

    public static BusinessFactory getInstance() throws Exception {
        if (factory == null) {
            factory = new BusinessFactory();
        }

        return factory;
    }

    public Business getBusiness() {
        return business;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    public IDeviceWriter getWriter() {
        return writer;
    }
}
```

这个简单的 BusinessFactory 将从 *config.properties* 文件中读取 *business.class* 与 *writer.class* 设置，并自动建立 Business 对 IDeviceWriter 实现的依赖。假设 *config.properties* 这样设置：

```
business.class=onlyfun.caterpillar.Business
writer.class=onlyfun.caterpillar.FloppyWriter
```

在程序中，可以按照下面的方式，通过 BusinessFactory 取得 Business 实例，并执行其 save()方法：

```
business = BusinessFactory.getInstance().getBusiness();
business.save();
```

根据 *config.properties* 的设置，Business 执行 save()方法时，实际上所使用的将是 FloppyWriter 实例的 saveToDevice()方法，如果 *config.properties* 中的 *writer.class* 换成了 *onlyfun.caterpillar.UsbDiskWriter*，则在以上的程序片段中，Business 在执行 save()方法时，实际上所使用的是 UsbDiskWriter 实例的 saveToDevice()方法。

通过一个简单的 BusinessFactory 实践，可以将对象的建立、依赖关系的建立隔离至应用程序的主要组件之外，事实上，Spring 核心容器就提供这样的配置管理功能，而且功能更为强大。在第 2 章将可以看到如何使用 Spring 的 IoC 容器功能，来建立 Business 的依赖关系。

良葛格的话匣子<<<

BusinessFactory 完整的程序示范，可以参考光盘中的 BusinessFactoryDemo 项目。

从以上的介绍来看，Dependency Inversion 的意思是“程序不依赖于实现，但是程序与实现都要依赖于抽象”。

IoC 的 Control 是控制的意思，其实背后的意义也是一种依赖关系的转移。如果 A 依赖于 B，其意义即 B 拥有控制权。如果转移这种关系（依赖关系的反转即控制关系的反转），将控制权由实现的一方转移至抽象的一方，让抽象方拥有控制权，可以获得组件的可重用性。在上面的 Business 程序中，整个控制权从实际的 FloppyWriter 转移至抽象的 IDeviceWriter

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接口上，让 Business 依赖于 IDeviceWriter 接口，且 FloppyWriter、UsbDiskWriter 也依赖于 IDeviceWriter 接口。

程序的业务逻辑部分应该要设计为可以重用的，不应受到所使用框架或容器的影响，这样将来才有可能转移整个应用程序的业务逻辑到其他的框架或容器上，如果业务逻辑过于依赖容器，则转移至其他的框架或容器时，就会发生困难。IoC 在容器中扮演的角色，可以用这么一句好莱坞名言来表示："Don't call me, I'll call you"。以程序的术语来说，就是"不要向容器要求所需要的（对象）资源，容器会自动将这些对象给您"。

IoC 的要求是容器不应该（或尽量不要）侵入应用程序，也就是不应出现与容器相依的 API。应用程序本身可以依赖于抽象的接口，容器可以通过这些抽象接口将所需的资源注入至应用程序中。应用程序不向容器主动要求资源，故而不会依赖于容器特定的 API，应用程序本身不会意识到正被容器使用，故而可以随时从容器系统中脱离，转移至其他的容器或框架而不需要作任何的修改。

1.3 依赖注入（Dependency Injection）

IoC模式基本上是一个高层的模式概念，在Martin Fowler的Inversion of Control Containers and the Dependency Injection pattern（*http://www.martinfowler.com/articles/injection.html*）中谈到，实现 IoC 有两种方式：Dependency Injection 与 Service Locator，Spring 所采用的是用 Dependency Injection 来实现 IoC，中文翻译为依赖注入。

依赖注入的意义是："保留抽象接口，让组件（Component）依赖于抽象接口，当组件要与其他实际的对象发生依赖关系时，由抽象接口来注入依赖的实际对象。"

依赖注入在 Martin Fowler 的文章中谈到了 3 种基本实现方式：Interface injection、Setter injection 与 Constructor injection。并分别称其为 Type 1 IoC、Type 2 IoC 与 Type 3 IoC。

- Type 2 IoC、Type 3 IoC

在前一个小节中所提到的 Business 实现的是 Type 2 IoC，通过 Setter（也就是 setXXX 方法）注入所依赖的对象，而 Type 3 IoC，则是在构造方法（Constructor）上注入依赖关系，例如：

```
public class BusinessObject {
   private IDeviceWriter writer;

   public BusinessObject(IDeviceWriter writer) {
      this.writer = writer;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }
    public void save() {
        ....
        writer.saveToDevice();
    }
}
```

Spring 鼓励使用 Setter injection（也就是 Type 2 IoC），但也允许您使用 Constructor injection（Type 3 Ioc）。要使用 Setter 或是 Constructor 来完成依赖关系注入，完全视您的需求而定，使用构造方法注入依赖对象的好处之一是，可以在构造对象的同时，一并建立依赖关系，然而如果要建立的对象关系很多，则必须在构造方法上声明一长串的参数。这时使用 Setter 来注入依赖关系会是个不错的选择，因为 Setter 有明确的方法名称可用于了解注入对象的作用或类型，使用 `setXXX()` 这样的名称，在构造方法上，会比必须记忆或查询某个参数位置所代表的对象要来得好。

- Type 1 IoC

Type 1 IoC 是 Interface injection，使用 Type 1 IoC 时会要求实现接口，对象所在的容器也会使用这个接口，容器知道接口上所规定的方法，所以可调用实现接口的对象来完成依赖关系的注入，例如容器的 API 中声明一个 `IDependency`：

```
public interface IDependency {
    public void createDependency(Map dependObjects);
}
```

接着，让 `Business` 类实现 `IDependency` 接口：

```
public class Business implement IDependency {
    private Map dependObjects; // 用以记录所依赖的对象

    public void createDependency(Map dependObjects) {
        this.dependObject = dependObjects;
        // 在这里实现与 Business 的依赖关系
        ......
    }

    public void save() {
        ....
        writer.saveToDevice();
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果要实现依赖关系注入的对象，必须实现 IDependency 接口，并让容器来管理对象，容器会执行被管理对象的 createDependency() 方法来建立依赖关系。

在上面的例子中，由于 Type 1 IoC 要求 Business 实现容器所规定的接口，这就使得 Business 依赖于容器的 API，如果日后 Business 打算脱离目前的这个容器到其他的容器或框架之中，就必须修改程序（也就是修改实现 IDependency 接口的程序部分）。这样的实现方式，在更复杂的依赖关系中将产生更多复杂的接口，组件与容器（框架）的依赖也会更加复杂，最后会使得组件无法从容器中脱离。

所以 Type 1 IoC 对应用程序或组件来说具有较强的侵入性，使用它来实现依赖注入，会使应用程序或组件相依于容器（框架），因而降低了组件的重用性。

Spring 的核心是个 IoC 容器，可以管理对象的生命周期，并可以通过 Setter 或构造函数的方式，编写配置文件（一个 XML 文件或是一个 *.properties* 文件），让 Spring 在执行时期根据配置文件的设定，为您建立对象之间的依赖关系。您不必特意编写一些程序来自行建立这些对象之间的依赖关系（如之前的 BusinessFactory），这不仅减少了大量的程序编写，降低了对象之间的耦合程度，通过 Spring 这类成熟的框架，还可以大幅提升应用程序的可靠性。

1.4 接下来的主题

关于入门 Spring 所需认识的术语及概念就先谈到这边，从下一个章节开始，将直接从 Spring 的设定与使用中，体会各个在这个章节介绍到的概念，首先将介绍如何在 Eclipse 中编写第一个 Spring 程序，并了解如何在 Eclipse 中使用 Spring IDE。



图 1.2 不要调用容器，容器会给您所想要的对象

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 1 章 Spring 入门

Spring 是个全方位的应用程序框架（Application framework），要了解 Spring 的所有功能需要相当长的时间，不过要完成第一个 Spring 程序却是相当简单的。在这个章节中，将实际完成第一个 Spring 程序，以增强您对使用 Spring 的信心，并使用 Spring 的 IoC 容器功能，管理第 1 章所介绍的 Business 对象之依赖关系。

在本书中将使用 Eclipse 作为集成开发环境（Integrated Development Environment, IDE），这是多数使用 Spring 开发程序的设计人员的选择，而这个章节中也将介绍如何在 Eclipse 中安装 Spring IDE，并了解 Spring IDE 的一些基本功能。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

2.1　第一个 Spring 程序

作为一个应用程序框架，入门第一个 Spring 程序是很简单的，这一个小节中将进行 Spring 的相关文件下载，并使用 Eclipse 进行设置、编写、执行，完成第一个 Spring 程序。

2.1.1　下载、设置 Spring

编写此书的时候，Spring 最新的版本是 2.0，请连接至 Spring 的官方网站进行文件下载，网址是 *http://springframework.org/*（如图 2.1 所示）。

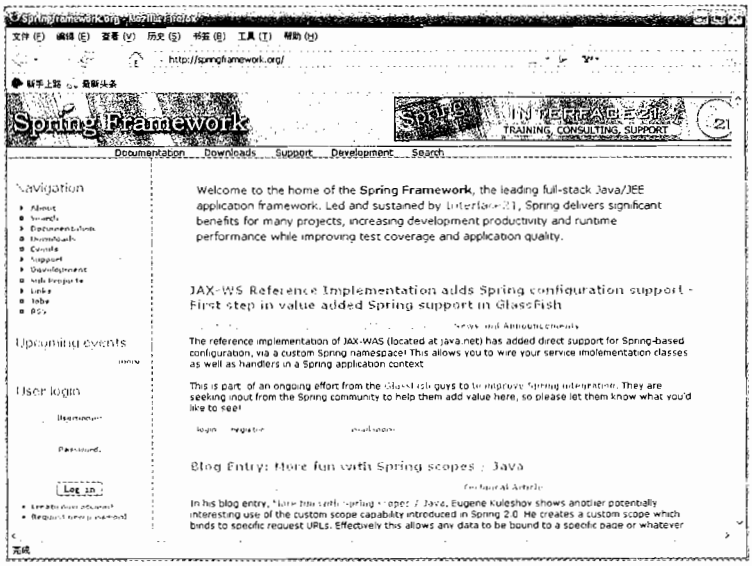


图 2.1　Spring 官方网址

点击左边的“Downloads”链接，可以进入下载页面，下载页面 Spring Framework 2.0 中会有个“Download”链接连接至 Sourceforge（*http://sourceforge.net/projectshowfiles.php?group_id=73357&package_id=173644&release_id=452461*）（如图 2.2 所示）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

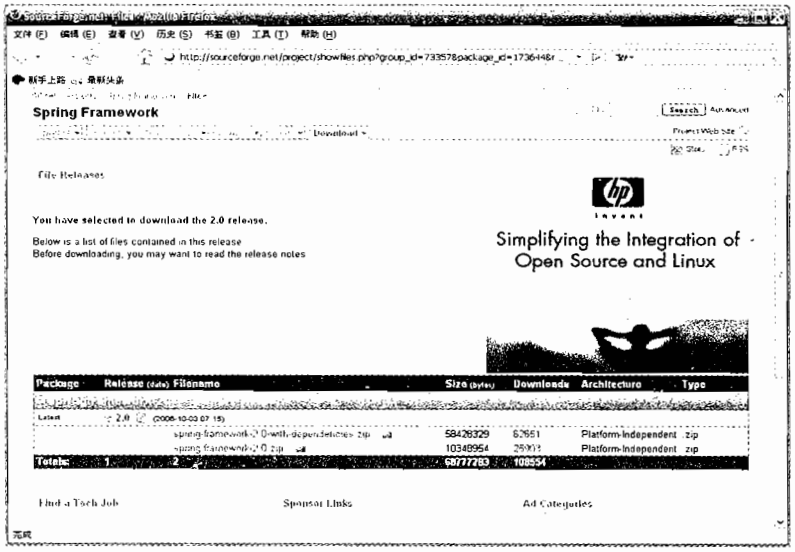


图 2.2 从 Sourceforge 下载 Spring 相关文件

Spring 在 Sourceforge 上提供了两个下载文件，一个是 *spring-framework-2.0-with-dependencies.zip*，一个是 *spring-framework-2.0.zip*。with-dependencies 的下载文件包括了一些 ant、jakarta-commons、struts、velocity 等与其他 Java 项目的开放源代码相依赖的 library 文件，如果您也需要这些相关文件，可以下载这个版本，如此就不用再分别到各个专门的网站下载这些文件了，如果已经有这些相关的文件，则只需要下载 *spring-framework-2.0.zip* 这个文件。

建议您直接下载 *spring-framework-2.0-with-dependencies.zip*，在选择文件的下载链接之后，会需要选择镜射（mirror）网站，可以选择一个最接近您所在区域的镜射网站，下载时会比较快一些。

下载 zip 文件并解压缩之后（如图 2.3 所示），在 *dist* 目录下的文件就是使用 Spring 时所需要的相关文件，如果下载的是 with-dependencies 版本，则在 *lib* 目录中的是可能会用到的相依赖的文件，如果需要查询 Spring 的 API 文件，或者需要阅读 Spring 参考手册，则可从 *docs* 目录中找到相关文件。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

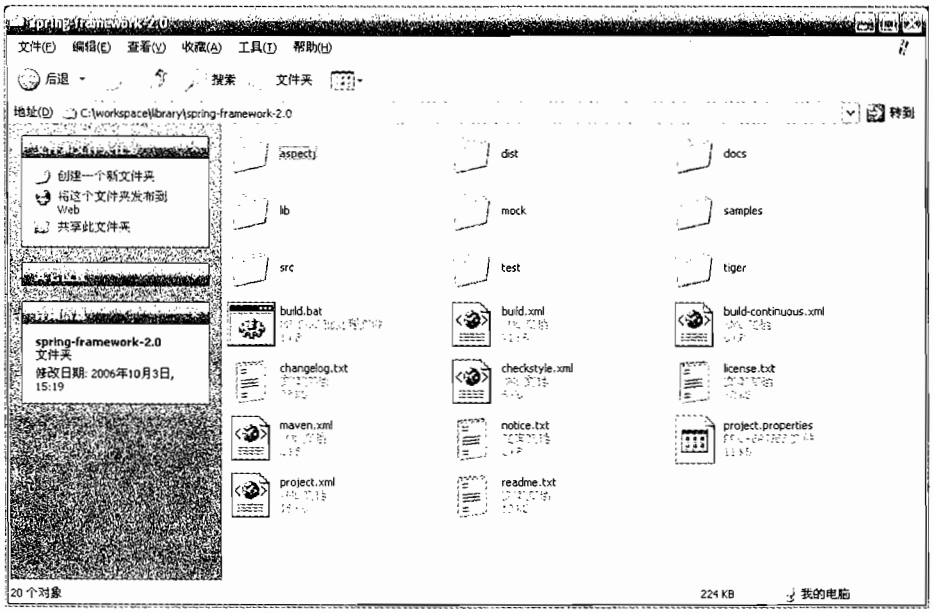


图 2.3　spring-framework-2.0-with-dependencies 的内容

在 *dist* 目录下，*spring.jar* 包含了所有 Spring 支持的功能所需要的类。如果需要的只是 Spring 核心，则可以在 *dist/modules* 目录下找到 *spring-core.jar*，如果日后需要用到 Spring 其他的子框架支持，再将其他的 *jar* 文件加入至 Classpath 的设置即可，例如 *spring-aop.jar*、*spring-webmvc.jar* 等。通常会直接使用 *spring.jar* 这个文件，好处是不需要加入个别的 *jar* 文件。

良葛格的话匣子<<<

为了统一操作的环境，假设您的 spring 相关文件都是放在 *C:\workspace\library\spring-framework-2.0* 中的，可参考图 2.3 的内容。另外所使用的 JDK 是 5.0，使用的集成开发环境是 Eclipse 3.2，可以在 Eclipse 官方网站 *http://www.eclipse.org/* 进行下载。使用 Eclipse 的原因是可以接触到编写 Spring 时所要注意的一些细节，当然，许多开发人员选择 Eclipse 来开发 Spring 应用程序也是原因之一。

2.1.2　准备 Spring 设计环境（使用 Eclipse）

在使用应用程序框架编写程序时，最好是搭配使用集成开发环境（Integrated Development Environment, IDE）以方便进行各种资源（如 Classpath）的管理，这里假设您已经下载了 Eclipse 3.2。请在执行了 Eclipse 之后先进行项目的新增，在执行了 Eclipse 菜单上的“File/New/Project...”

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

项目之后，在“New Project”对话框中选择“Java Project”，点击“Next”按钮，并为您的项目取名称，这里假设命名为 SpringDemo，且项目是储存在 *C:\workspace* 目录下的，如图 2.4 所示。

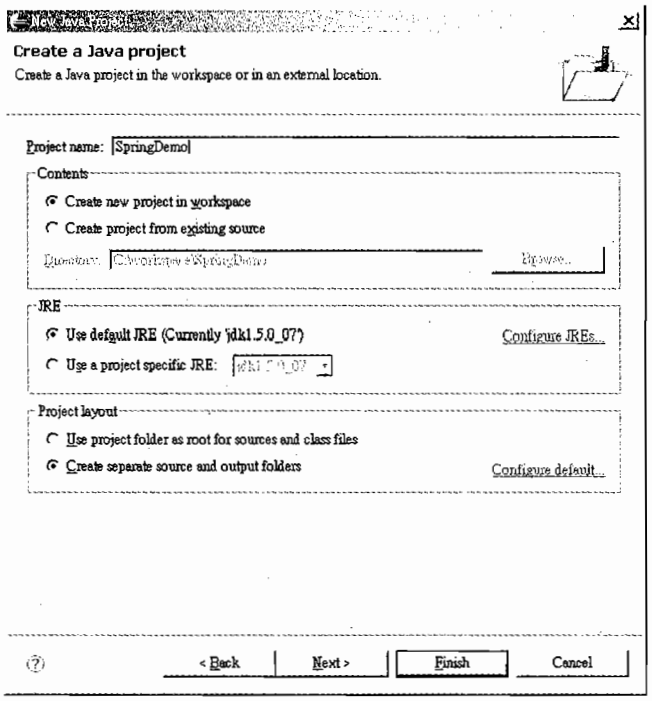


图 2.4 在 Eclipse 中新建 SpringDemo 项目

建议将*.java* 原始程序代码跟编译好的*.class* 文件分开放置，这可以在图 2.4 的“Project Layout”中设置，选择“Create separate source and output folders”，预设*.java* 会放置在项目的 *src* 目录下，而*.class* 会放置在项目的 *bin* 目录下。

在新建项目之后，可以在项目指定的目录中，直接按鼠标右键新增各种文件，以后各章节中若提到新建某文件，都是这样进行操作，如图 2.5 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

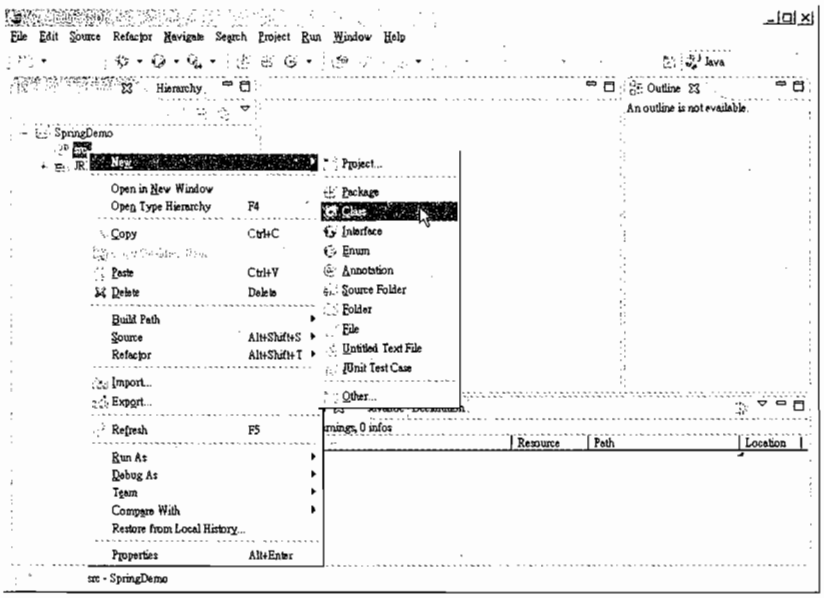


图 2.5　在 Eclipse 中新建文件的方式

接着，必须将与 Spring 相关的 library 加入到项目管理之中（具体而言就是将 *.jar* 文件设置到 Classpath 中），请执行菜单上的“Project/Properties”项目，选择“Properties for SpringDemo”对话框上的“Java Build Path”，并切换页面标签至“Libraries”，如图 2.6 所示。

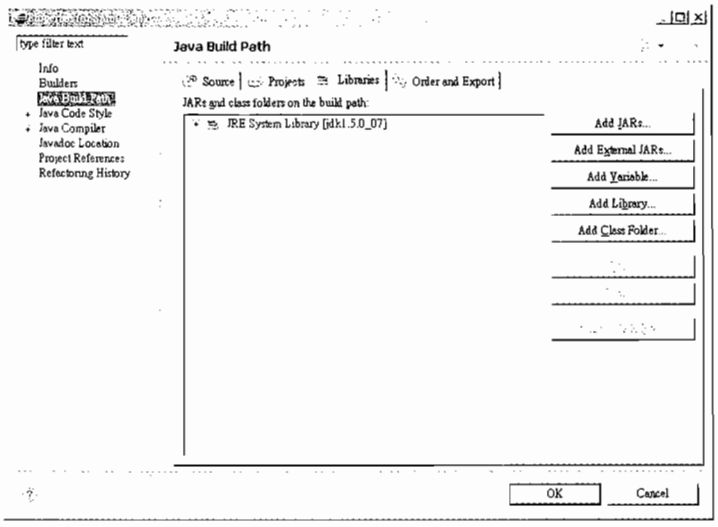


图 2.6　设置 library

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以在这个页面中加入相依赖的*jar*文件，例如点击“Add External JARS...”，即可选择 *.jar* 文件的来源，不过为了未来设置 Spring 的 library 方便起见，您可以在 Eclipse 中自定义“User Library”。请点击“Add Library...”按钮，在接下来的“Add Library”窗口中选择“User Library”后点击“Next”按钮，并点击“User Libraries...”按钮，接着将会显示图 2.7 的窗口。在此窗口中点击“New...”按钮，可以新增 User Library，请在“New User Library”窗口中取名“Spring 2.0 Core”后点击“OK”按钮。

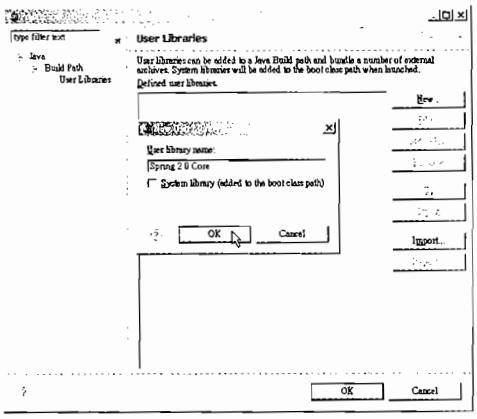


图 2.7 新增 User Librariy

接着可以点击“Add JARs...”按钮，将 *spring-core.jar*、*spring-beans.jar*，以及相依赖的 *commons-logging.jar* 加入，这是稍后示范的第一个 Spring 程序所需的*.jar* 文件，可以在 *dist/modules* 目录及 *lib/jakarta-commons* 目录中找到这些文件，另外若想要使用 Log4j 来设置日志（logging）消息，则还需要加入 *lib/log4j* 目录下的 *log4j-1.2.14.jar*。

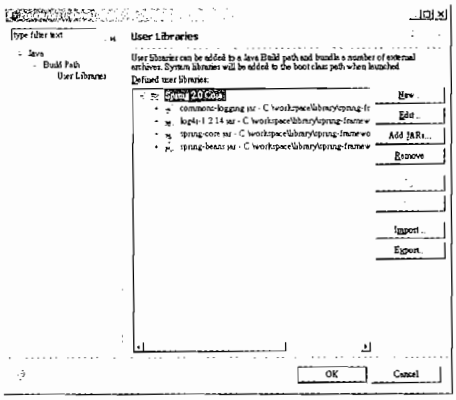


图 2.8 新增 Spring 2.0 Core Library

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

完成以上步骤后，点击“OK”按钮即可完成自定义的 User Library 的新增如图 2.8 所示，之后若需要加入自定义的 Library，则只要在图 2.6 的“Java Build Path”中选择“Add Library...”，并选择“User Library”即可，如图 2.9 所示。

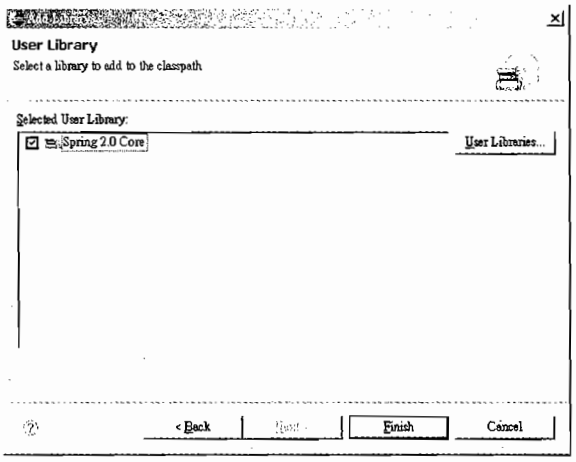


图 2.9　加入 User Library

2.1.3　编写第一个 Spring 程序

现在来编写您的第一个组件（Component），它只是一个简单的 JavaBean，用来向用户打声招呼：

SpringDemo　　HelloBean.java

```
package onlyfun.caterpillar;

public class HelloBean {
    private String helloWord;

    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }

    public String getHelloWord() {
        return helloWord;
    }
}
```

稍后您可以通过 setHelloWord()这个 Setter 来设置新的招呼语（为强调注入的是文字，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

所以使用 Word），不过并不是亲自编写程序来做这些事，而是在配置文件中加以定义，由 Spring 来为您做设置的动作，接着可以编写 Bean 的定义文件，定义文件会告诉 Spring 容器，如何完成相依赖对象的关系注入等动作，Bean 定义文件的文件名称可以自由定义，例如这里取名为 *beans-config.xml*：

SpringDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean">
        <property name="helloWord">
            <value>Hello!Justin!</value>
        </property>
    </bean>

</beans>
```

在 Bean 定义的文件中是以 <beans> 作为根节点的，使用 <bean> 来为每一个 Bean 进行设置。"id" 属性用以设置 Bean 实例的识别名称，稍后可以使用"id"所指定的名称来取得 Bean 的实例，"class" 属性用来指定 Bean 的类名称，<property> 标签的"helloWord"设置了 Setter 的名称（setHelloWord），并在 <value> 标签上设置了将要注入的字符串值。

接着来编写一个简单的示范程序：

SpringDemo　　SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;

public class SpringDemo {
    public static void main(String[] args) {
        Resource rs =
            new ClassPathResource("beans-config.xml");
        BeanFactory factory = new XmlBeanFactory(rs);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        HelloBean hello =
            (HelloBean) factory.getBean("helloBean");
        System.out.println(hello.getHelloWord());
    }
}
```

org.springframework.core.io.ClassPathResource 类实现了 org.springframework.core.io.Resource 接口，可从 *Classpath* 下读取指定的 Bean 定义文件，如果要指定 Bean 定义文件的绝对路径或相对路径位置，则可以使用 org.springframework.core.io.FileSystemResource。

由于这里使用的是 XML 定义文件，所以使用 org.springframework.beans.factory.xml. XmlBeanFactory 类，XmlBeanFactory 实现了 org.springframework.beans.factory.xml.BeanFactory 接口，它被用来读取定义并建立 BeanFactory 实例。具体来说，BeanFactory 就是 Spring 所提供的 IoC 容器。

就如同第 1 章中所实现的 BusinessFactory 类的功能，Spring 的 BeanFactory 也是 Factory 模式的一个实现例子，但更具通用性，可以建立、管理不同类型的对象。要注意的是，在 Spring 1.2 之后，XmlBeanFactory 只接受实现 Resource 接口的对象，如 ClassPathResource、FileSystemResource、InputStreamResource、ServletContextResource、UrlResource 等。

这是从较低层次的角度来使用 Spring 的 IoC 容器功能，通过 BeanFactory 来读取配置文件，并完成依赖的关系注入。这里的依赖是什么？指的是 HelloBean 的实例依赖于 String 对象，通过 Setter 所保留的接口，使用 Setter injection 来完成这个依赖注入，而不是将招呼语写在 HelloBean 中。

BeanFactory 读取 Bean 的配置设置并完成关系维护之后，可以通过 getBean()方法指定 Bean 的名称来取得 Bean 实例。如果使用 BeanFactory，只有在真正需要 Bean 对象时，才会建立实际的 Bean 实例，而不是在一开始建立 BeanFactory 时就建立 Bean 实例。

Spring 预设会在主控台下显示许多日志消息，为了方便观看执行程序的结果，可以设置日志消息的显示层级，因此可以在项目的 *Classpath* 下建立 *log4j.properties* 文件，内容如下所示：

SpringDemo　　log4j.properties

```
log4j.rootLogger=WARN, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这个文件中，日志的显示级别设置只有出现警示（WARN）消息时，它才会显示在标准输出（stdout）之中。在 Eclipse 中，以上所编写的各文件之相对位置如图 2.10 所示。

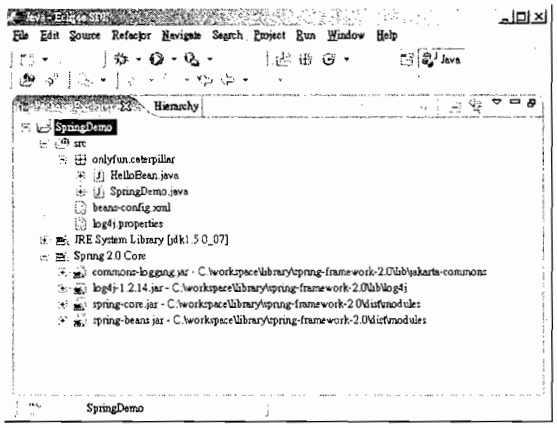


图 2.10 第一个 Spring 程序

可以在开启 *SpringDemo.java* 文件后，执行菜单上的“Run/Run As/Java Application”来看看实际运行之后的结果（如图 2.11 所示）。



图 2.11 第一个 Spring 程序执行结果

如果想要改变招呼语，只需要更改 *beans-config.xml* 中`"helloWord"`属性的设置就可以了，不用修改 *HelloBean.java*。从通用的角度来看，这意味着，如果想要改变一些对象之间的依赖关系，只要修改配置文件即可，不用修改组件的任何一行程序代码。

良葛格的话匣子<<<

在光盘的 examples 目录下找到相对应的 SpringDemo 项目，可以将之复制至 workspace 目录，解除只读属性，然后在开启 Eclipse 项目时，将项目名称取为与 SpringDemo 相同的名称，Eclipse 会自动判断项目的文件结构并完成项目设置。

2.1.4 编写第二个 Spring 程序

先前所编写的第一个 Spring 程序是很简单的程序，主要目的是为了让您了解完成一个简

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

单的 Spring 程序所需的步骤。接下来回顾一下第 1 章所示范的 Business 程序，当时是自行编写一个 BusinessFactory 类，负责读取定义文件、生成对象并建立依赖关系。在这里可以改用 Spring 所提供的功能，以 BeanFactory 取代 BusinessFactory。

为了完整地呈现项目内容，下面将所需的相关文件一并列出，首先程序依赖于抽象的 IDeviceWriter 接口：

SpringDemo2 IDeviceWriter.java

```
package onlyfun.caterpillar;

public interface IDeviceWriter {
    public void saveToDevice();
}
```

有个 FloppyWriter 实现了 IDeviceWriter 接口，如下所示：

SpringDemo2 FloppyWriter.java

```
package onlyfun.caterpillar;

public class FloppyWriter implements IDeviceWriter {
    public void saveToDevice() {
        System.out.println("储存至软盘...");
    }
}
```

假设 Business 类有个储存资料的需求，它依赖于抽象的 IDeviceWriter 接口，实际上储存资料的行为依赖于 IDeviceWriter 的实现对象，Business 类实现如下：

SpringDemo2 Business.java

```
package onlyfun.caterpillar;

public class Business {
    private IDeviceWriter writer;

    public void setDeviceWriter(IDeviceWriter writer) {
        this.writer = writer;
    }

    public IDeviceWriter getDeviceWriter() {
        return writer;
    }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void save() {
        if(writer == null) {
            throw new RuntimeException("DeviceWriter needed...");
        }
        writer.saveToDevice();
    }
}
```

在第 1 章是自行编写一个 `BusinessFactory` 类，并设置一个 *config.properties* 进行类信息的相关设置。现在可以直接使用 Spring 所提供的 IoC 容器功能来设置 Bean 定义文件，如下所示：

SpringDemo2 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="writer" class="onlyfun.caterpillar.FloppyWriter"/>

    <bean id="business"
          class="onlyfun.caterpillar.Business">
        <property name="deviceWriter">
            <ref bean="writer"/>
        </property>
    </bean>
</beans>
```

在定义文件中设置了两个 Bean 的信息，由于程序只需产生 `FloppyWriter` 实例，所以不用给它设置任何的依赖关系。直接使用`<bean>`标签设置一个`"id"`为 writer 的 Bean 实例，并将之给 `Business` 的实例引用。要引用 IoC 容器中已设置的 Bean 实例，可以使用`<ref>`标签，并指定它的`"bean"`属性为所依赖的 Bean 实例的`"id"`名称，如此 IoC 容器会自动建立两个对象之间的依赖关系。

您还可以建立 *log4j.properties* 文件，内容与前一个项目相同即可，之后的范例将不再重复说明 *log4j.properties* 的设置。最后，请建立一个示范程序，如下所示：

SpringDemo2 SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.core.io.ClassPathResource;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.core.io.Resource;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;

public class SpringDemo {
    public static void main(String[] args) {
        Resource rs =
            new ClassPathResource("beans-config.xml");
        BeanFactory factory = new XmlBeanFactory(rs);

        Business business =
            (Business) factory.getBean("business");
        business.save();
    }
}
```

程序执行结果的一个参考画面如图 2.12 所示。



图 2.12　第二个 Spring 程序执行结果

假设现在 Business 类不再将资料储存至软盘，而是储存在 USB 移动硬盘上，则可以让一个 UsbDiskWriter 类实现 IDeviceWriter 接口，并在 Bean 定义文件中修改"writer"这个 Bean 的"class"属性为 UsbDiskWriter 类。如此重新执行程序，Business 类底层所依赖的储存行为就会替换为 UsbDiskWriter 所定义的储存行为，而不用对 Business 作出任何的修改。

2.1.5　Spring 2.0 的 XML 设置

如果您曾经使用过 Spring 1.2 或之前的版本，也许已经发现，在这里所示范的 XML 定义文件中，XML 设置的语法是基于 XML Schema 的：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    ...
</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果想要使用 Spring 2.0 所提供的一些新功能来对 XML 标签进行设置，建议采用基于 XML Schema 的设置方式，Spring 开发团队也建议采用 XML Schema，结合 Spring 2.0 的新标签进行设置，这样可以让 XML 配置文件的内容更简洁且容易读取。

如果不打算或无法采用基于 XML Schema 的设置方式，Spring 2.0 中也提供了基于 DTD 的设置方式：

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
   "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
   ...
</beans>
```

采用基于 XML Schema 的方式进行设置是被鼓励的，假若想要使用 Spring 2.0 新的<util>标签设置，只需要如下设置：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/util
  http://www.springframework.org/schema/util/spring-util-2.0.xsd">
   ...
</beans>
```

对于 Spring 2.0 新增的一些标签的设置方式，在往后适当的章节中，将会分别加以说明。

2.2 安装、使用 Spring IDE

不免仍是老话一句：“工欲善其事，必先利其器”。使用 Eclipse 并结合 Spring 来开发应用程序是个不错的组合，但可以更好，可以在 Eclipse 中安装 Spring IDE 的 plug-in，让您在使用 Spring 开发应用程序时有更高的效率，本节将介绍如何安装与使用 Spring IDE。

2.2.1 安装 Spring IDE

Spring IDE 是 Spring 官方网站所推荐的 Eclipse plug-in，在开发 Spring 时可提供对 Bean 定义文件进行验证、以可视化的方式观看 Bean 与 Bean 之间的依赖关系等功能，下面来介绍一下如何在 Eclipse 中安装 Spring IDE。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在安装 Spring IDE 之前，可以先安装 Graphical Editing Framework，因为 Spring IDE 以它为基础显示 Bean Graph，也就是以图形化的方式显示 Bean 与 Bean 之间的依赖关系。首先请执行 Eclipse 菜单上的“Help/Software updates/Find and Install”，然后选择“Search for new features to install”，再点击“Next”按钮，如图 2.13 所示。

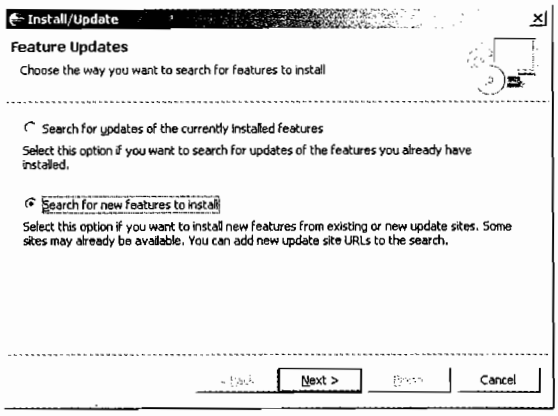


图 2.13　搜寻可用的 Plug-in

在接下来的“Install”窗口中，点击“New Remote Site...”按钮，新增 Spring IDE 的 Update Site，网址是：*http://springide.org/updatesite/*如图 2.14 所示。

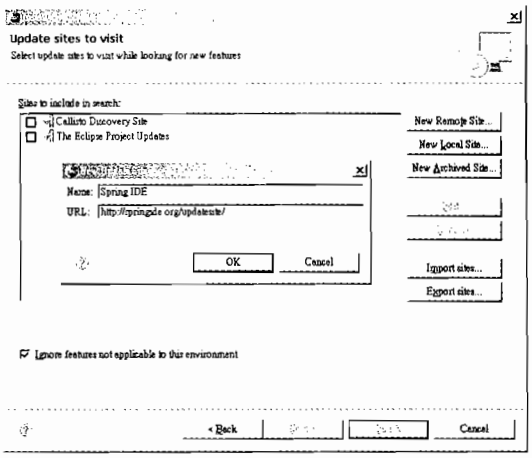


图 2.14　新增 Spring IDE 的 Update Site

在新增了 Spring IDE 的 Update Site 之后，请选择新增的“Spring IDE”项目，且一并选择“Callisto Discovery Site”，以便安装 Graphical Editing Framework。点击“Finish”按钮后

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

还需要选择“Update Site Mirrors”，请在选择了“Callisto Discovery Site”后点击“OK”按钮，接着会搜导 plug-in。在编写这个章节时，Graphical Editing Framework 的版本为 3.2.1，Spring IDE 最新的版本为 1.3.6，请按图 2.15 的选择项目进行安装。

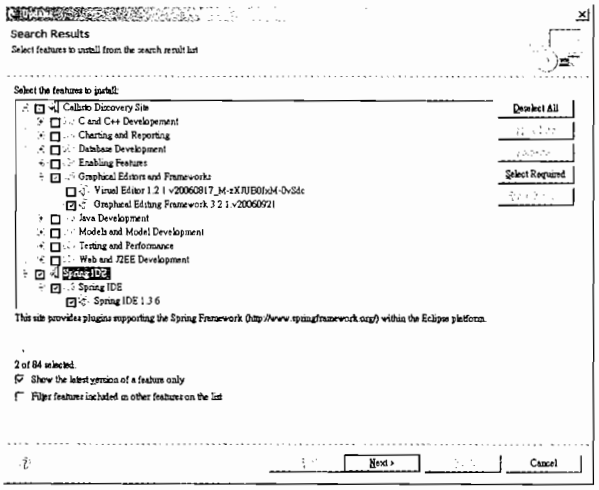


图 2.15 安装 Graphical Editing Framework 与 Spring IDE

接下来请根据安装精灵的指引，一步一步完成 Graphical Editing Framework 与 Spring IDE 的下载与安装，安装完成后会要求重新启动 Eclipse，重新启动之后，可以执行菜单上的“Help/About Eclipse SDK”指令，并点击“Plug-in Details”来看看 Graphical Editing Framework 与 Spring IDE 是否安装完全（如图 2.16 所示）。

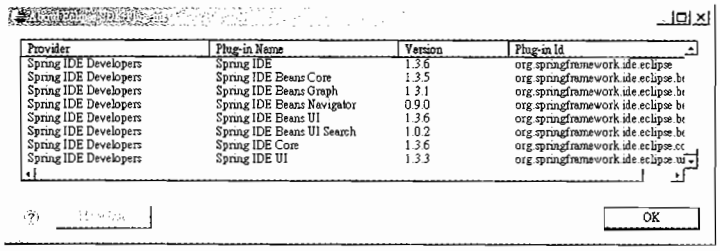


图 2.16 检视 Graphical Editing Framework 与 Spring IDE 是否安装完全

2.2.2 使用 Spring IDE

以先前所开发的 SpringDemo 来作示范，要加入 Spring IDE 的功能支持，请在 SpringDemo 项目上按鼠标右键，执行“Add Spring Project Nature”指令，如图 2.17 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

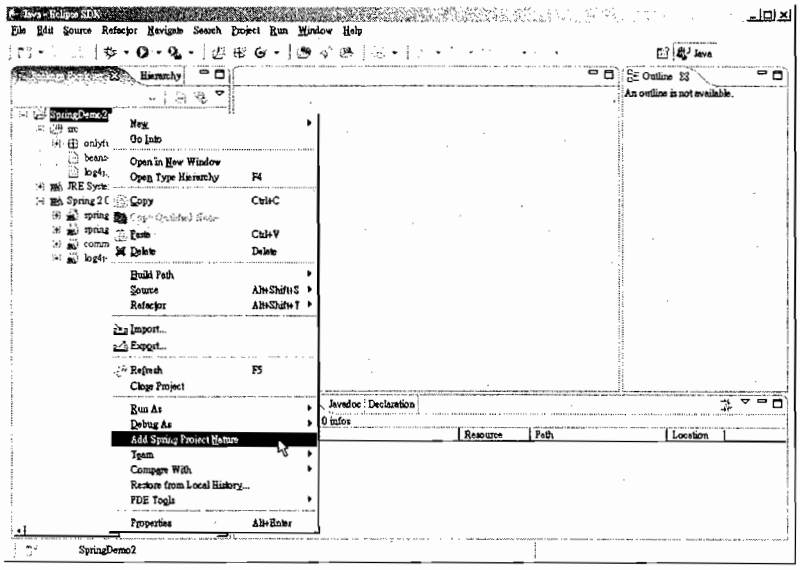


图 2.17　在项目上加入 Spring IDE 支持

加入 Spring IDE 的支持后，项目图示上会出现一个“S”字样。Spring IDE 可以协助检查 Bean 定义文件的内容设置是否正确。为了拥有这个功能，必须先执行菜单列上的“Project/Properties”指令，然后选择“Spring”项目，在“Config Files”页签中点击“Add...”按钮，加入要进行验证管理的 Bean 定义文件，如图 2.18 所示。

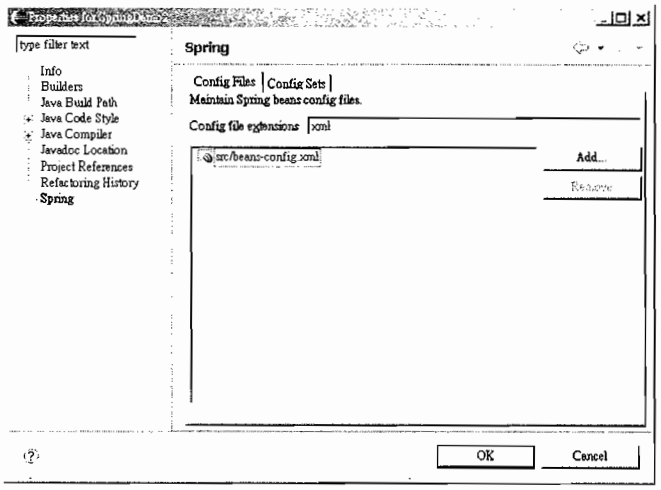


图 2.18　将 Bean 定义文件纳入验证管理

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Spring IDE 除了可以验证 XML 文件中的标签如 <beans> 等是否设置正确之外，还可以协助验证所设置的类名称是否正确，例如若设置时键入了错误的类名称，则会出现错误标识与说明（如图 2.19 所示）。

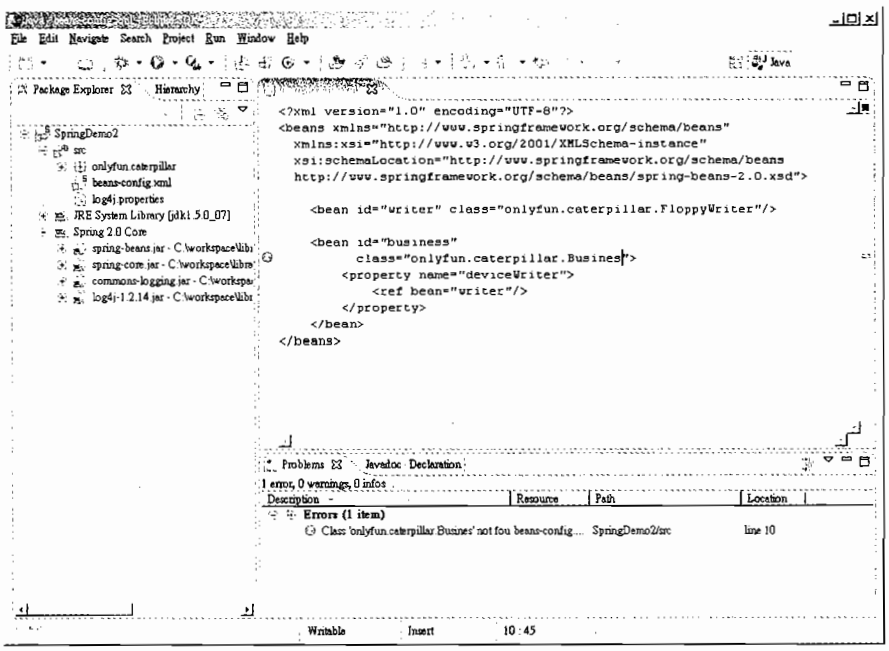


图 2.19 Spring IDE 可检验出 Bean 定义文件上的错误设置

切换 Eclipse 右下方的页签至“Spring Beans”（如果没有出现，请执行菜单上的“Window/Show View/Other...”，并选择“Spring/Spring Beans”），Spring IDE 可以在这个页签中以可视化的方式显示 Bean 定义文件中的节点摘要（如图 2.20 所示）。

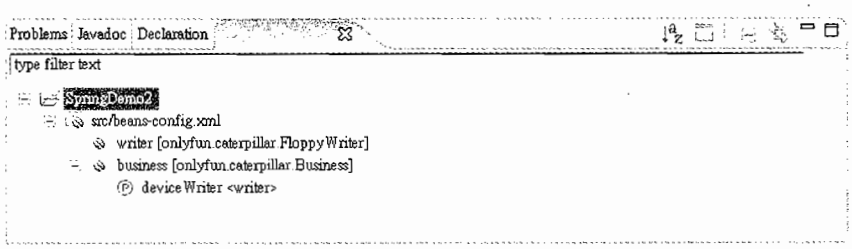


图 2.20 Spring IDE 的 Beans 节点摘要显示

在 Bean 定义文件上按鼠标右键执行“Show Graph”指令，Spring IDE 会以图形化的方式，显示 Bean 定义文件中 Bean 的定义，以及 Bean 与 Bean 之间的依赖关系（如图 2.21

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

所示）。

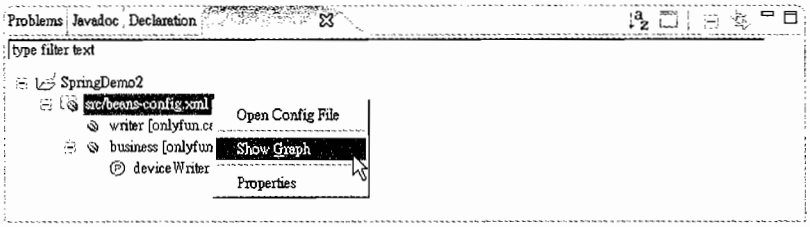


图 2.21　显示 Bean Graph

以先前所编写的 SpringDemo2 项目为例，Spring IDE 将会显示如图 2.22 所示的 Bean Graph。

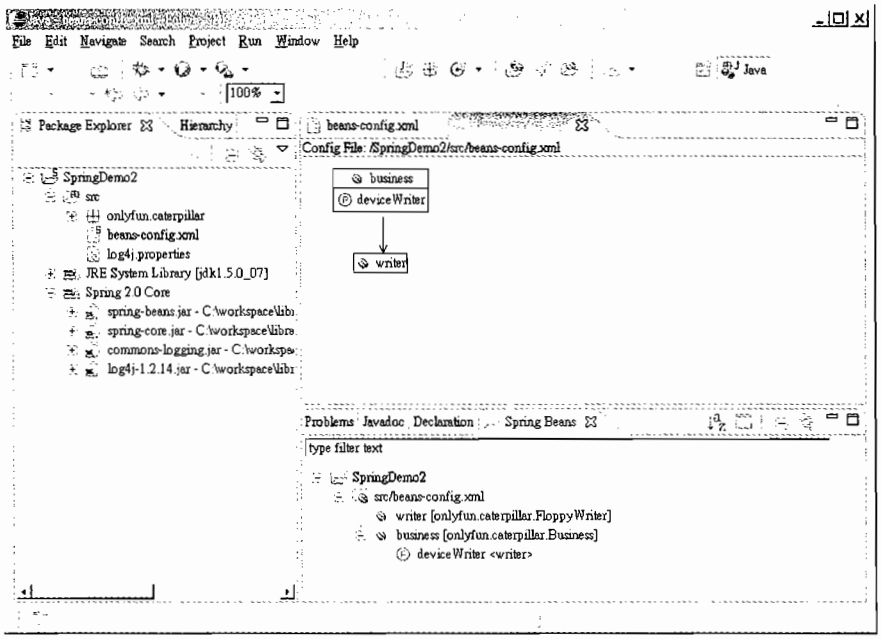


图 2.22　Spring IDE 的可视化 Bean 管理

2.3　接下来的主题

上节中示范了两个简单的 Spring 程序，让您了解了如何在 Eclipse 中开发 Spring 应用程序，并介绍了 Spring IDE 的入门使用。入门 Spring 是简单的，但接下来就要一步一步地探索 Spring 的每一个功能了，从下一个章节开始，将接续这个章节的第一个 Spring 程序，深入了

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

解 Spring 的 IoC 容器的功能与设置方式。



图 2.23 您的第一个 Spring

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第

Bean、消息、事件

章

Spring 的核心是一个容器（Container），它实现了 IoC 的概念，可以协助管理各个对象的生命周期，以及对象之间的依赖关系。在核心容器的使用上，熟悉 BeanFactory 与 ApplicationContext 的运用是了解 Spring 的重点所在，在这一章节中，您将可以了解如何使用 Spring 容器进行各种 Bean 的配置与管理。

另一方面，作为一个应用程序框架（Application framework），ApplicationContext 除了具备 BeanFactory 基本的容器管理功能之外，还能支持更多应用程序框架的特性，像资源的取得、文字消息的解析、事件的处理与传播等功能。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.1　Bean 基本管理

在第 2 章中已经实际完成了第一个 Spring 程序，并大致了解了何谓“依赖注入”（Dependency Injection），如何使用 Spring 的容器功能来管理 Bean。在这节中，您将会学习到更多有关于 Bean 在 Spring 中的设置方式，以及生命周期。

3.1.1　BeanFactory、ApplicationContext

BeanFactory 负责读取 Bean 定义文件；管理对象的加载、生成；维护 Bean 对象与 Bean 对象之间的依赖关系；负责 Bean 的生命周期。对于简单的应用程序来说，使用 BeanFactory 就已经足够来管理 Bean 了，在对象的管理上已可以获得许多的方便性，BeanFactory 接口包括了 6 种方法可以调用。

- boolean containsBean(String)

 测试 BeanFactory 中是否包含指定名称的 Bean。

- Object getBean(String)

 指定 Bean 定义文件中设置的名称，可取得相对应的 Bean 实例。

- Object getBean(String, Class)

 指定 Bean 定义文件中设置的名称，取得相对应的 Bean 实例，并转换（Cast）至指定的类。

- Class getType(String name)

 指定 Bean 定义文件中设置的名称，取得相对应的 Bean 的 Class 实例。

- boolean isSingleton(String)

 指定 Bean 定义文件中设置的名称，测试指定的 Bean 之 scope 是否为 Singleton（之后在 Bean 的 scope 中会说明）。

- String[] getAliases(String)

 指定 Bean 定义文件中设置的名称，取得该 Bean 所有的别名（之后在 Bean 的识别名称与别名中会说明）。

不过作为一个应用程序框架，只提供 Bean 容器管理的功能是不够的，若要利

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

用 Spring 所提供的一些特色以及高级的容器功能，可以使用 org.springframework.context.ApplicationContext。ApplicationContext 是基于 BeanFactory 而建立的，也具备读取 Bean 定义文件，维护 Bean 之间的依赖关系等功能，除此之外，ApplicationContext 还提供一个应用程序所需的更完整的框架功能，例如：

- 提供更方便地取得资源文件（Resource file）的方法。
- 提供解析文字消息的方法。
- 支持国际化（Internationalization, I18N）消息。
- ApplicationContext 可以发布事件，对事件感兴趣的 Bean 可以接收到这些事件。

Spring 的创始者 Rod Johnson 建议使用 ApplicationContext 来取代 BeanFactory，在实现 ApplicationContext 的类中，最常使用的大概是以下 3 个。

- org.springframework.context.support.FileSystemXmlApplicationContext

 可指定 XML 定义文件的相对路径或绝对路径读取定义文件。

- org.springframework.context.support.ClassPathXmlApplicationContext

 从 *Classpath* 设置路径中读取 XML 定义文件。

- org.springframework.web.context.support.XmlWebApplicationContext

 在 Web 应用程序的文件架构中，指定相对位置读取定义文件。

举个简单的例子，可以将第 2 章中完成的第一个 Spring 程序中的 SpringDemo 类修改为以下的内容：

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
          ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                        "beans-config.xml");
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        HelloBean hello =
            (HelloBean) context.getBean("helloBean");
        System.out.println(hello.getHelloWord());
    }
}
```

ApplicationContext 可以读取多个 Bean 定义文件，可以在实例化 ApplicationContext 的实现类时，以数组指定 Bean 定义文件的位置，例如：

```
ApplicationContext context = new ClassPathXmlApplicationContext(
        new String[] {"beans-config.xml", "beans-config2.xml"});
```

您可以使用 *file:/*、*classpath:*甚至 *http://*等 URL 前置，或者 *classpath**：表示所有的 *Classpath* 前置路径都匹配：

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("classpath*:beans-config.xml");
```

也可以指定*字符，例如以下的例子可读取 *Classpath* 下所有以"beans"开头的 XML 配置文件，但要注意的是此方法只在实际的文件系统中有用，如果是在.*jar* 文件中，以下的指定就无效：

```
ApplicationContext context =
    new ClassPathXmlApplicationContext("beans*.xml");
```

如果您先前只将 *spring-beans.jar* 与 *spring-core.jar* 加入至 *Classpath* 中，为了能编译与执行以上的程序，还必须将 *spring-context.jar* 也加入至 *Classpath* 中。

当需要多个 Bean 定义文件时，Spring 开发团队建议使用 ApplicationContext 的方式来读取，好处是 Bean 定义文件之间是各自独立的，不用意识到彼此的存在。另一个替代的方式是使用<import>标签，例如：

```
<beans ...>
    <import resource="dao-config.xml"/>
    <import resource="resources/messageSource.xml"/>

    <bean id="bean1" class="..."/>
    <bean id="bean2" class="..."/>
</beans>
```

<import>标签必须放置在<bean>标签之前，定义文件必须放置在同一个目录或是 *Classpath* 之中，以相对路径指定 Bean 定义文件位置，而每个定义文件的内容都必须包括<beans>根标签。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

上面介绍了在编写程序时，如何使用 ApplicationContext 替换 BeanFactory。关于 ApplicationContext 对文字消息的管理、事件的发布等功能，也将在往后相关的主题中一一介绍。

3.1.2 Bean 的识别名称与别名

在定义文件中使用<bean>标签时可以使用"id"指定 Bean 的识别名称，当您需要多个 Bean 定义文件时，最好规范"id"名称的命名方式，以免名称上的冲突。在设置了"id"属性之后，您还可以为 Bean 设置别名，例如考虑在 A 定义文件中要参考一个"device:dataSource"的 Bean 实例，而在 B 定义文件中要参考一个"user:dataSource"的 Bean 实例，但实际上 DataSource 的实例在应用程序中只存在一个，这就可以采用设置别名的方式为该实例指定"device:dataSource"与"user:dataSource"的名称，例如：

```
<beans ...>
    <bean id="app:dataSource" class="..."/>
    <alias name="app:dataSource" alias="device:dataSource"/>
    <alias name="app:dataSource" alias="user:dataSource"/>
    ...
</beans>
```

在 A 定义文件中，可以直接参考别名，例如：

```
<beans ...>
    <bean id="device:someBean" class="...">
        <property>
            <ref bean="device:dataSource">
        </property>
    </bean>
    ...
</beans>
```

在 B 定义文件中，也可以参考另一个别名，例如：

```
<beans ...>
    <bean id="user:otherBean" class="...">
        <property>
            <ref bean="user:dataSource">
        </property>
    </bean>
    ...
</beans>
```

两个定义文件可以拥有自己的名称设置规范，而实际上参考的是同一个 Bean 实例，除了

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用<alias>标签设置别名之外，还可以直接使用<bean>标签的"name"属性来设置别名，多个别名之间以逗号隔开，例如：

```
<beans ...>
   <bean id="app:dataSource"
         name="device:dataSource,user:dataSoource" class="..."/>
   ...
</beans>
```

3.1.3 Bean 的实例化

在 Spring 中要实例化一个 Bean 可以有几种方式，到目前为止所看到的是最基本的方式：

```
<bean id="writer" class="onlyfun.caterpillar.FloppyWriter"/>
```

依这样的设置方式，Spring 将会使用默认的构造方法，也就是没有参数的构造方法来建立 Bean 实例（关于有参数的构造方法的设置，下一节将会说明）。

在设计上的一种方式是通过静态工厂方法（static factory method）来取得某个对象，好处是调用静态工厂方法的对象不用了解对象建立的细节，例如可以设计一个 IMusicBox 接口：

StaticFactoryMethodDemo　　IMusicBox.java

```
package onlyfun.caterpillar;

public interface IMusicBox {
   public void play();
}
```

假设您设计了一个 MusicBoxFactory，取得 IBox 实例的细节由它的 createMusicBox() 静态方法负责：

StaticFactoryMethodDemo　　MusicBoxFactory.java

```
package onlyfun.caterpillar;

public class MusicBoxFactory {
   public static IMusicBox createMusicBox() {
      return new IMusicBox() {
         public void play() {
            System.out.println("播放钢琴音乐...");
         }
      };
   }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在 Spring 中也可以使用设置静态工厂的方法来取得 Bean 实例，例如若要通过 MusicBoxFactory 的 createMusicBox() 方法来取得 IMusicBox 的实例，则可以设置<bean>的"factory-method"属性，例如：

StaticFactoryMethodDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="musicBox"
         class="onlyfun.caterpillar.MusicBoxFactory"
         factory-method="createMusicBox"/>
</beans>
```

接下来可以编写一个简单的测试程序，看看以上的设置是否正确取得 IMusicBox 的实例：

StaticFactoryMethodDemo　　SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
                    ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("beans-config.xml");
        IMusicBox musicBox = (IMusicBox) context.getBean("musicBox");
        musicBox.play();
    }
}
```

执行的结果如下图 3.1 所示。



图 3.1　StaticFactoryMethodDemo 项目执行结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

与这个范例类似的，如果您想要使用某个工厂 Bean 实例的工厂方法（factory method）来取得 Bean 实例，则可以使用<bean>的"factory-bean"属性指定工厂 Bean，并使用"factory-metod"属性指定工厂方法来取得 Bean 实例，例如设置一个如下的范例片段：

```
<!-工厂 Bean，其上有 createInstance()方法 -->
<bean id="factoryBean" class="onlyfun.caterpillar.SomeFactory">
  ...
</bean>

<bean id="some"
      factory-bean="factoryBean"
      factory-method="createInstance"/>
```

在以上的设置中，Spring 将会实例化"factoryBean"，而要取得"some"的实例时，将会使用"factoryBean"的 createInstance()方法。

3.1.4 Bean 的 scope

在 Spring 中，从 BeanFactory 或 ApplicationContext 取得的实例被默认为 Singleton，也就是默认每一个 Bean 名称只维持一个实例，例如在第 2 章 SpringDemo 项目中，每一次通过 factory.getBean ("helloBean") 取得的对象，实际上都是同一个对象，而不是每一次都产生一个新的对象。

使用 Singleton 模式产生单一实例，对单线程的程序来说并不会有什么问题，但对于多线程的程序，您必须注意到线程安全（Thread-safe）的问题，防止多个线程同时存取共用资源所引发的资料不同步问题，通常 Singleton 的 Bean 都是无状态的（Stateless）。

然而在 Spring 中可以设置，每次从 BeanFactory 或 ApplicationContext 指定别名并取得 Bean 时都产生一个新的实例，例如：

```
...
<bean id="helloBean"
        class="onlyfun.caterpillar.HelloBean"
        scope="prototype">
...
```

在 Spring 中，"scope" 属性预设是 "singleton"，通过将其设置为 "prototype"，使得每次指定名称来取得 Bean 时，都会产生一个新的实例。您也可以通过设置<bean>的"singleton"属性为"true"或"false"，来设置是否以 Singleton 的方式产生实例，这种设置"singleton"属性的方式，主要是为了与先前版本的方式兼容而保留的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

要注意到，Spring 的 Singleton 主要是针对“一个 IoC 容器维持一个 Bean 实例”而言的。与设计模式（Design Pattern）上常谈到的 Singleton 不同，设计模式上谈到的 Singleton，是指对每个 ClassLoader 所载入的类产生一个实例。

在 Spring 2.0 中，"scope"属性除了可以设置"singleton"与"prototype"之外，针对 Web 应用程序环境，还可以设置"request"、"session"与"globalSession"，分别表示请求阶段、会话阶段与基于 Portlet 的 Web 应用程序会话阶段。

3.1.5 Bean 的生命周期

一个 Bean 从建立到销毁，会历经几个执行阶段，如果使用 BeanFactory 来生成、管理 Bean，会尽量支持以下的生命周期。

- **Bean 的建立**

 由 BeanFactory 读取 Bean 定义文件，并生成各个 Bean 实例。

- **属性注入**

 执行相关的 Bean 属性依赖注入。

- **BeanNameAware 的 setBeanName()**

 如果 Bean 类有实现 org.springframework.beans.factory.BeanNameAware 接口，则执行它的 setBeanName() 方法。

- **BeanFactoryAware 的 setBeanFactory()**

 如果 Bean 类有实现 org.springframework.beans.factory.BeanFactoryAware 接口，则执行它的 setBeanFactory() 方法。

- **BeanPostProcessors 的 processBeforeInitialization()**

 如果有任何的 org.springframework.beans.factory.config. BeanPostProcessors 实例与 Bean 实例关联，则执行 BeanPostProcessors 实例的 processBeforeInitialization() 方法。

- **InitializingBean 的 afterPropertiesSet()**

 如果 Bean 类有实现 org.springframework.beans.factory.InitializingBean，则执行它的 afterPropertiesSet() 方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- Bean 定义文件中定义 init-method

 可以在 Bean 定义文件使用 "init-method" 属性设置方法名称，例如：

```
...
<bean id="helloBean"
        class="onlyfun.caterpillar.HelloBean"
        init-method="initBean">
...
```

 如果有以上设置，当代码运行至这个阶段时，就会执行 initBean() 方法。

- BeanPostProcessors 的 processaAfterInitialization()

 如果有任何的 BeanPostProcessors 实例与 Bean 实例关联，则执行 BeanPostProcessors 实例的 processaAfterInitialization() 方法。

- DisposableBean 的 destroy()

 在容器关闭时，如果 Bean 类有实现 org.springframework.beans.factory.DisposableBean 接口，则执行它的 destroy() 方法。

- Bean 定义文件中定义 destroy-method

 在容器关闭时，可以在 Bean 定义文件使用 "destroy-method" 属性设置方法名称，例如：

```
...
<bean id="helloBean"
     class="onlyfun.caterpillar.HelloBean"
     destroy-method="destroyBean">
...
```

 如果有以上设置，当代码运行至这个阶段时，就会执行 destroyBean() 方法。

 定义<bean>的"init-method"属性，其实与实现 InitializingBean 的 afterPropertiesSet()意义是相同的。采用前者，可以定义 Bean 的初始方法，而又不用耦合至 Spring 的 API。定义<bean>的"destroy-method"属性，与实现 DisposableBean 接口的 destroy() 方法的意义也是相同的。采用前者，Bean 不用耦合至 Spring 的 API 里。

如果所有的 Bean 都有固定的初始名称或销毁名称，例如都命名为 init()或 destroy()，则不必在<bean>上分别定义"init-method"与"destroy-method"属性，但可以在<beans>

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

上定义"default-init-method"与"default-destroy-method"属性，例如：

```
<beans default-init-method="init"
       default-destroy-method="destroy">
        ....
</beans>
```

如以上的设置之后，Spring 会自动执行每个 Bean 上所定义的 init()方法与 destroy()方法，可以省去不少编写配置文件的功夫。

如果使用 ApplicationContext 来生成并管理 Bean 则稍有不同。使用 ApplicationContext 来生成及管理 Bean 实例，在执行 BeanFactoryAware 的 setBeanFactory() 阶段之后，若 Bean 类上有实现 org.springframework.context. ApplicationContextAware 接口，则执行其 setApplicationContext() 方法，接着才继续进行 BeanPostProcessors 的 processBeforeInitialization() 及之后的流程。

在非 Web 应用程序中，若想在关闭容器之前调用 Bean 定义文件中 Singleton 的 Bean 所设置的 destroy 方法，则可以执行 AbstractApplicationContext 的 registerShutdownHook()方法，并向 JVM 注册相关方法，例如：

```
AbstractApplicationContext context
        = new ClassPathXmlApplicationContext("beans-config.xml");
...
context.registerShutdownHook();
// 执行应用程序...
```

在应用程序结束之前，Bean 定义文件上所设置的 destroy 方法将会被调用执行。

如果使用 BeanFactory，只有在使用 getBean()方法真正取得 Bean 时，才会做实例化的动作。如果使用 ApplicationContext，则会预先针对 Bean 定义文件的内容，将所有的 Bean 实例化。如果这不是您所想要的，则可以在<bean>上设置"lazy-init"属性为"true"，ApplicationContext 就不会在启动时针对该 Bean 作实例化的动作，例如：

```
<bean id="some" class="onlyfun.caterpillar.Some" lazy-init="true">
    ...
</bean>
```

3.1.6 Bean 定义的继承

如果 Bean 定义文件的内容不断地增长，而您发现有些 Bean 的定义其实有所重复。例如，也许好几个 Bean 定义都有"name"与"age"等属性，且大部分设置都是相同的值，只有几个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Bean 会有不同的设置，则可以考虑继承某个 Bean 定义，这样可以省去许多设置的功夫。

举个实际的例子来示范 Bean 定义的继承，假设您编写了一个如下的 SomeBean 类。

BeanInheritanceDemo SomeBean.java

```
package onlyfun.caterpillar;

public class SomeBean {
    private String name;
    private int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

您也许会产生许多 SomeBean 的实例，其中大部分 SomeBean 的"name"与"age"属性都是"guest"与 18，若只有几个不同，则可以如下定义。

BeanInheritanceDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="inheritedSomeBean" abstract="true">
        <property name="name">
            <value>guest</value>
        </property>
        <property name="age">
            <value>18</value>
        </property>
    </bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="some"
          class="onlyfun.caterpillar.SomeBean"
          parent="inheritedSomeBean">
        <property name="name">
            <value>Justin</value>
        </property>
    </bean>
</beans>
```

在"inheritedSomeBean"中设置了"abstract"属性为"true"，表示这是个抽象的 Bean 定义，Spring 不会尝试去实例化，而在"some"中设置了"parent"属性，表示它将继承"inheritedSomeBean"的设置，只有"name"属性会重新定义为"Justin"。可以写个简单的程序来看看"some"这个 Bean 最后的属性设置。

BeanInheritanceDemo　　SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
                    ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("beans-config.xml");
        SomeBean some = (SomeBean) context.getBean("some");
        System.out.println("name:" + some.getName());
        System.out.println("age:" + some.getAge());
    }
}
```

可以在执行结果中看到，"name"被重新定义为"Justin"，而"age"则继承了"inheritedSomeBean"，也就是 18（如图 3.2 所示）。

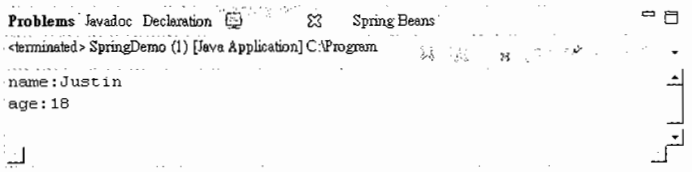


图 3.2　BeanInheritanceDemo 项目执行结果

除了从一个完全抽象的 Bean 定义继承相关设置之外，还可以从一个 Bean 实例的定义来

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

继承，例如：

```
...
    <bean id="inheritedSomeBean" class="onlyfun.caterpillar.SomeBean">
        <property name="name">
            <value>guest</value>
        </property>
        <property name="age">
            <value>18</value>
        </property>
    </bean>

    <bean id="some"
          class="onlyfun.caterpillar.SomeBean"
          parent="inheritedSomeBean">
        <property name="name">
            <value>Justin</value>
        </property>
    </bean>
```

在这个设置中，"inheritedSomeBean"也可以被 Spring 实例化，若有必要，还可以被其他 Bean 继承其定义。

3.2 Bean 的依赖设置

在 Spring 中两种基本的依赖注入方式是 Setter Injection 及 Constructor Injection，另外针对非 Singleton 的依赖注入，还提供了 Method Injection。在这节中，将针对这些依赖注入，以及一些依赖关系设置的方式加以说明。

3.2.1 Type 2 IoC、Type 3 IoC

在第 2 章中完成的第一个 Spring 程序中，利用了 Bean 的 Setter 方法完成依赖注入。Spring 鼓励的是使用 Setter Injection 完成依赖注入，也就是 Type 2 Dependency Injection，但也允许使用 Type 3 Dependency Injection 的 Constructor injection，使用 Setter 或 Constructor 来注入依赖关系要视需求而定，这在稍后再加以讨论，这里先来看看如何在 Spring 中使用 Constructor injection，首先来看看 HelloBean 类如何编写：

Type3Demo　　HelloBean.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class HelloBean {
    private String name;
    private String helloWord;

    // 建议要有无参数构造方法
    public HelloBean() {
    }
    public HelloBean(String name, String helloWord) {
        this.name = name;
        this.helloWord = helloWord;
    }

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }
    public String getHelloWord() {
        return helloWord;
    }
}
```

在定义 Bean 类时，为了能让 Spring 有可以使用无参数构造方法来生成对象的弹性，建议可以定义一个无参数的构造方法，即使目前没有编写任何的实现内容。

在 HelloBean 类定义中要注意的是，第二个有参数的构造方法上两个参数的顺序，Bean 定义文件中使用 Constructor Injection 时，在设置上可以依构造方法上参数的顺序来指定，如下所示。

Type3Demo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <constructor-arg>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            <value>Justin</value>
        </constructor-arg>
        <constructor-arg>
            <value>Hello</value>
        </constructor-arg>
    </bean>

</beans>
```

在定义文件中，使用 <constructor-arg> 标签来表示将使用 Constructor Injection，由于使用 Constructor Injection 并不像使用 Setter Injection 时拥有 setXXX() 这样易懂的名称，所以指定时必须依照参数的顺序。另一个方式是指定索引位置，"index"属性就是用于指定对象将注入至构造方法中哪一个位置的参数，参数的顺序指定中，第一个参数的索引值是 0，第二个是 1，依此类推，例如 *beans-config.xml* 中也可以这么编写：

```
...
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <constructor-arg index="0">
            <value>Justin</value>
        </constructor-arg>
        <constructor-arg index="1">
            <value>Hello</value>
        </constructor-arg>
    </bean>
...
```

接着是编写主程序，使用 ApplicationContext 读取定义文件内容，生成 Bean 实例，完成依赖关系注入，程序的编写如下所示：

Type3Demo　　SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
                    ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                    "beans-config.xml");

        HelloBean hello =
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            (HelloBean) context.getBean("helloBean");

        System.out.print("Name: ");
        System.out.println(hello.getName());
        System.out.print("Word: ");
        System.out.println(hello.getHelloWord());
    }
}
```

执行结果如图 3.3 所示。

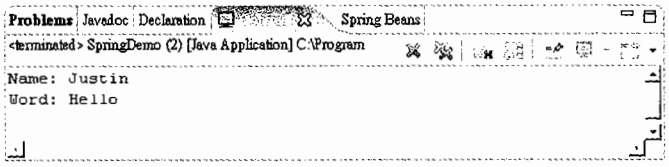


图 3.3 Type3Demo 项目执行结果

当构造方法上的参数个数相同时，Spring 会自动解析构造方法上的参数类型及所设置的依赖注入，用以决定要使用哪个构造方法，例如：

```
...
public class HelloBean {
    ...
    public HelloBean(String name) {
        ...
    }
    public HelloBean(Date date) {
        ...
    }
    ...
}
```

HelloBean 在构造时可以指定 String，也可以指定 java.util.Date 对象，如果在配置文件中是这么设置的：

```
...
    <bean id="date" class="java.util.Date"/>

    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean">
        <constructor-arg>
            <ref bean="date"/>
        </constructor-arg>
    </bean>
...
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

由于配置文件中是注入 Date 类型的对象，则 Spring 会自动解析并使用有 Date 类型参数的构造方法，用它来构造对象并注入所依赖的 Date 对象，您也可以使用"type"属性来明确指定要使用哪个类型，例如：

```
...
   <bean id="date" class="java.util.Date"/>

   <bean id="helloBean"
        class="onlyfun.caterpillar.HelloBean">
      <constructor-arg type="java.util.Date">
         <ref bean="date"/>
      </constructor-arg>
   </bean>
...
```

至于是应该使用 Constructor 方法还是 Setter 方法来完成依赖注入，这就像是在讨论一个古老的问题：是要在对象建立时就准备好所有的资源，还是应在对象建立好后，再使用 Setter 方法来进行设置。

使用 Constructor 的好处之一是，可以在构造对象的同时一并完成依赖关系的建立，对象建立后，它与其他对象的依赖关系也就准备好了，但如果要建立的对象关系很多，使用 Constructor injection 会在构造方法上留下一长串的参数，且不易记忆，这时使用 Setter 方法会是个不错的选择。

另一方面，Setter 方法具有明确的方法名称，通过它可以了解注入的对象是什么，像记忆 setXXX() 这样的名称，会比记忆 Constructor 上某个参数位置的索引代表某个对象来得好，当结合 IDE 的方法提示功能使用时，编写程序会更方便且有效率。

然而使用 Setter 方法时，由于提供的是 setXXX() 方法，所以不能保证相关的资料成员或资源在执行时不会被更改设置，因为程序开发人员可能会直接执行 Setter 方法来设置相关属性，所以如果想要让一些资料成员或资源变为唯读或是私有的，使用 Constructor injection 会是个简单的选择。

3.2.2　依赖的值设置与参考

之前的例子在定义 Bean 时，可以在<property>或<constructor>中直接使用<value>标签，用它来指定一个字符串或基本类型（primitive）给属性或构造方法，例如：

```
...
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <constructor-arg>
            <value>Justin</value>
        </constructor-arg>
        <property name="age">
            <value>18</value>
        </property>
    </bean>
...
```

您也可以直接使用"value"属性来指定字符串或基本类型值，这是一种比较简洁的设置方式，例如：

```
...
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <constructor-arg value="Justin"/>
        <property name="age" value="18"/>
    </bean>
...
```

如果想要设置某个属性为 null 值，可以使用<null/>标签，例如：

```
...
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <property name="name"><null/></property>
    </bean>
...
```

注意以下的设置方式，是将字符串属性设置为空字符串""，而不是设置为 null：

```
...
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <property name="name"><value></value></property>
    </bean>
...
```

如果在 Bean 定义文件中已经有一个定义的 Bean 实例，则可以直接让某个属性参考至这个实例，例如：

```
...
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <constructor-arg>
            <ref bean="date"/>
        </constructor-arg>
        <property name="other">
            <ref bean="otherBean"/>
        </property>
    </bean>
...
```

另一个比较简洁的写法，是使用"ref"属性来指定，例如：

```
...
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <constructor-arg ref="date"/>
        <property name="other" ref="otherBean"/>
    </bean>
...
```

如果希望使用<ref>参考其他 Bean 实例时，所定义的 Bean 必须是在同一个设置文件中，且可以指定"local"属性，例如：

```
...
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
        <property name="other">
            <ref local="otherBean"/>
        </property>
    </bean>
...
```

如果某个 Bean 实例只被某个属性参考过一次，之后在定义文件中不再被其他 Bean 属性参考，那么也可以直接在属性定义时使用 <bean> 标签，并仅需指定其 "class" 属性即可，例如：

```
...
    <bean id="helloBean" class="onlyfun.caterpillar.HelloBean">
        <property name="helloWord">
            <value>Hello!</value>
        </property>
        <property name="date">
            <bean class="java.util.Date"/>
        </property>
    </bean>
...
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Spring 的 IoC 容器会自动生成 Date 实例，并通过 setDate()方法将 Date 实例设置给"helloBean"。

在取得某个 Bean 之前，如果它依赖于另一个 Bean，Spring 就会先去实例化被依赖的 Bean 并进行依赖注入。如果某个 Bean 在生成之前要求另一个 Bean 必须先实例化，则可以指定"depends-on"属性来设置，例如：

```
<bean id="beanOne"
   class="onlyfun.caterpillar.BeanOne" depends-on="beanTwo"/>

<bean id="beanTwo" class=onlyfun.caterpillar.BeanTwo" />
```

在上例中，虽然"beanOne"并没有明确要求"beanTwo"注入，但由于设置了"depends-on"属性为"beanTwo"，则在取得"beanOne"之前，Spring 会先将"beanTwo"实例化，如果有两个以上的 Bean 要设置在"depends-on"中，则以逗号区隔。

3.2.3 自动绑定

除了在 Bean 定义文件中使用<value>指定字符串以及基本类型值，使用 <ref> 直接指定参考至其他 Bean 实例，或是使用 <bean> 标签指定 "class" 属性，用来指定依赖对象外，Spring 也支持隐式的自动绑定。您可以通过类型（byType）或名称（byName），将某个 Bean 实例绑定至其他 Bean 对应的属性上，这里直接以 AutoWireDemo 项目中的程序为例，示范各种绑定的方式，假设 HelloBean 类如下定义：

AutoWireDemo　　HelloBean.java

```
package onlyfun.caterpillar;

import java.util.Date;

public class HelloBean {
   private String helloWord;
   private Date date;

   public void setHelloWord(String helloWord) {
      this.helloWord = helloWord;
   }

   public String getHelloWord() {
      return helloWord;
   }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void setDate(Date date) {
        this.date = date;
    }

    public Date getDate() {
        return date;
    }
}
```

其中 HelloBean 的 setDate() 方法接受一个 Date 的实例，在以下的 Bean 定义文件中，将"autowire"属性设置为"byType"，让 Spring 自动依类型做依赖绑定。

AutoWireDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dateBean" class="java.util.Date"/>

    <bean id="helloBean" class="onlyfun.caterpillar.HelloBean"
          autowire="byType">
        <property name="helloWord" value="Hello!"/>
    </bean>

</beans>
```

在定义文件中，并没有指定"helloBean"的 Date 属性，而是通过自动绑定，在"autowire"属性上指定了"byType"，所以会根据"helloBean"的 setDate() 方法所接受的类型来判断在 Bean 定义文件中是否定义了类似的类型对象，并将之设置给了"helloBean"的 setDate()。使用自动绑定时，如果"byType"无法完成绑定，则丢出 org.springrframework.beans.factory.UnsatisfiedDependencyExcpetion 异常。可以编写一个简单的测试程序，看看依赖关系是否完成。

AutoWireDemo SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
                    ClassPathXmlApplicationContext;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("beans-config.xml");

        HelloBean hello =
            (HelloBean) context.getBean("helloBean");

        System.out.print("Word: ");
        System.out.println(hello.getHelloWord());
        System.out.print("Date: ");
        System.out.println(hello.getDate());
    }
}
```

执行结果参考画面如图 3.4 所示。

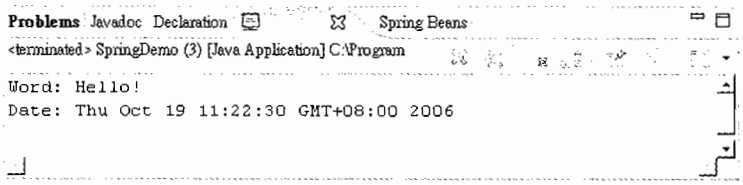


图 3.4 AutoWireDemo 项目执行结果

您也可以指定 "byName" 来绑定，则 Spring 会判断 Bean 定义时"id" 属性上指定的别名与 Setter 名称是否一致来进行自动绑定。举个例子来说，如果"byName"要通过 setDate() 方法来完成依赖注入，则必须修改第一个 Bean 的"id"值为 V 名称，例如修改 *beans-config.xml* 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="date" class="java.util.Date"/>

    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean"
         autowire="byName">
        <property name="helloWord" value="Hello!"/>
    </bean>
</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果使用 "byName" 无法完成自动绑定，则对应的 Setter 仅维持未绑定状态。

也可以在使用 Type 3 Dependency Injection 时套用自动绑定，也就是在构造方法上也可以尝试进行自动绑定，例如若修改 AutoWireDemo 项目中的 HelloBean 类，如下所示。

```
package onlyfun.caterpillar;

import java.util.Date;

public class HelloBean {
    private String helloWord;
    private Date date;

    // 建议保留一个无参数的构造方法
    public HelloBean() {
    }

    public HelloBean(Date date) {
        this.date = date;
    }

    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }

    public String getHelloWord() {
        return helloWord;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public Date getDate() {
        return date;
    }
}
```

然后重新定义 *beans-config.xml*，如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="date" class="java.util.Date"/>

    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean"
         autowire="constructor">
        <property name="helloWord" value="Hello!"/>
    </bean>
</beans>
```

由于 "autowire" 设置为 "constructor"，在建立依赖关系时，Spring 容器会试图对比容器中的 Bean 实例类型，以及相关的构造方法上的参数类型，若在类型上符合，则选用该构造方法来建立 Bean 实例。在上例中，"helloBean"的带参数构造方法与 date 这个 Bean 的类型相符，于是选用了该构造方法来构造实例，并将定义文件中的 date 实例注入给它，如果无法完成绑定，则丢出 org.springframework.beans.factory.UnsatisfiedDependencyException 异常。

可以看到，运用一些自动绑定的方式，可以缩短定义文件的编写内容，如果还想再偷懒的话，可以设置为 "autodetect"，一切交给 Spring 来判断，例如修改一下 *beans-config.xml*：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="date" class="java.util.Date"/>

    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean"
         autowire="autodetect">
        <property name="helloWord" value="Hello!"/>
    </bean>
</beans>
```

当 "autowire" 被设置为 "autodetect" 时，Spring 会尝试使用"autowire" 被设置为 "constructor" 来处理依赖关系的建立，如果没有完成依赖关系的建立，则再尝试使用 "autowire" 被设置为 "byType" 的方式来建立依赖关系。

在以上介绍的隐式自动绑定中，由于没办法从定义文件中清楚地看到是否每个属性都完

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

成了设置，为了确定某些依赖关系确实建立，您可以加入依赖检查，在 <bean> 标签使用时设置 "dependency-check"，可以有 4 种依赖检查方式："simple"、"objects"、"all"、"none"。

"simple" 只检查简单的属性是否完成依赖关系，像是原生（primitive）数据类型或字符串对象；"objects"则检查对象类型的属性是否完成依赖关系；"all"检查全部的属性是否完成依赖关系；"none"是默认值，表示不检查依赖性。

一个设置的例子如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="date" class="java.util.Date"/>

    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean"
         autowire="autodetect"
         dependency-check="all">
       <property name="helloWord" value="Hello!"/>
    </bean>
</beans>
```

一旦使用自动绑定时加入了依赖检查设置，如果进行依赖检查时发现有未完成的依赖关系，则执行程序时会丢出 org.springframework.beans.factory.UnsatisfiedDependencyException 异常。

由于使用自动绑定，在定义文件上不容易查看对象之间的依赖关系，因此建议只在应用程序需要快速开发原型或需求未确定时，才使用自动绑定的功能，以简化初期开发时琐碎的配置设置工作，在应用程序功能确定之后，再清楚地设置对象与对象间的依赖关系。

3.2.4 集合对象

对于数组、java.util.List、java.util.Set、java.util.Map 等集合对象，在注入前必须填充入一些对象至集合中，然后再将集合对象注入至所需的 Bean，也可以交由 Spring 的 IoC 容器来自动维护或生成集合对象，并完成依赖注入。

这里直接举个完整的应用程序作示范，例如有个 SomeBean 类定义，如下所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

CollectionDemo SomeBean.java

```
package onlyfun.caterpillar;

import java.util.List;
import java.util.Map;

public class SomeBean {
    private String[] someStrArray;
    private Some[] someObjArray;
    private List someList;
    private Map someMap;

    public String[] getSomeStrArray() {
        return someStrArray;
    }

    public void setSomeStrArray(String[] someStrArray) {
        this.someStrArray = someStrArray;
    }

    public Some[] getSomeObjArray() {
        return someObjArray;
    }

    public void setSomeObjArray(Some[] someObjArray) {
        this.someObjArray = someObjArray;
    }

    public List getSomeList() {
        return someList;
    }

    public void setSomeList(List someList) {
        this.someList = someList;
    }

    public Map getSomeMap() {
        return someMap;
    }

    public void setSomeMap(Map someMap) {
        this.someMap = someMap;
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在 SomeBean 类中只是简单地定义一些数组、List 与 Map 属性，稍后这些属性所需的依赖对象将由 Spring 来生成注入，在 SomeBean 中还使用到 Some 类，其内容如下：

CollectionDemo Some.java

```
package onlyfun.caterpillar;

public class Some {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String toString() {
        return name;
    }
}
```

对于数组或 List 类型的依赖关系注入，在编写定义文件时是使用 <list> 标签，并使用 <value> 标签指定字符串，或是使用 <ref> 来参考至其他的 Bean 实例；对于 Map 类型的依赖关系注入则是使用 <map> 标签，Map 必须指定 key-value，所以您要用 <entry> 标签指定 key，然后使用 <value> 标签指定字符串，或是使用 <ref> 来参考至其他的 Bean 实例。

CollectionDemo 项目的 *beans-config.xml* 定义编写如下：

CollectionDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="some1" class="onlyfun.caterpillar.Some">
        <property name="name" value="Justin"/>
    </bean>

    <bean id="some2" class="onlyfun.caterpillar.Some">
        <property name="name" value="momor"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    </bean>

    <bean id="someBean" class="onlyfun.caterpillar.SomeBean">
        <property name="someStrArray">
            <list>
                <value>Hello</value>
                <value>Welcome</value>
            </list>
        </property>

        <property name="someObjArray">
            <list>
                <ref bean="some1"/>
                <ref bean="some2"/>
            </list>
        </property>

        <property name="someList">
            <list>
                <value>ListTest</value>
                <ref bean="some1"/>
                <ref bean="some2"/>
            </list>
        </property>

        <property name="someMap">
            <map>
                <entry key="MapTest">
                    <value>Hello!Justin!</value>
                </entry>
                <entry key="someKey1">
                    <ref bean="some1"/>
                </entry>
            </map>
        </property>
    </bean>

</beans>
```

为了测试定义文件编写是否正确，以及依赖关系是否如预期的建立，您可以编写一个如下的测试程序。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

CollectionDemo　　SpringDemo.java

```
package onlyfun.caterpillar;

import java.util.List;
import java.util.Map;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
                  ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "beans-config.xml");

        SomeBean someBean =
            (SomeBean) context.getBean("someBean");

        // 取得数组类型依赖注入对象
        String[] strs =
            (String[]) someBean.getSomeStrArray();
        Some[] somes =
            (Some[]) someBean.getSomeObjArray();
        for(int i = 0; i < strs.length; i++) {
            System.out.println(strs[i] + ","
                  + somes[i].getName());
        }

        // 取得 List 类型依赖注入对象
        System.out.println();
        List someList = (List) someBean.getSomeList();
        for(int i = 0; i < someList.size(); i++) {
            System.out.println(someList.get(i));
        }

        // 取得 Map 类型依赖注入对象
        System.out.println();
        Map someMap = (Map) someBean.getSomeMap();
        System.out.println(someMap.get("MapTest"));
        System.out.println(someMap.get("someKey1"));
    }
}
```

执行的结果如图 3.5 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

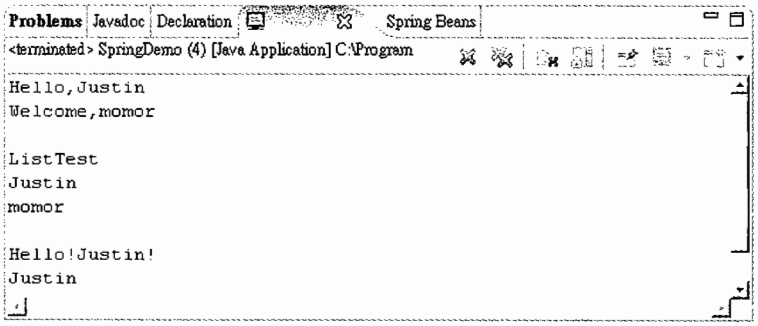


图 3.5 CollectionDemo 项目执行结果

在设置 Map 注入时，<entry>标签原本按以下的形式设置：

```
<entry key="MapTest">
   <value>Hello!Justin!</value>
</entry>
```

还可以编写为：

```
<entry key="MapTest" value="Hello!Justin!"/>
```

如果 key-value 分别要被参考至某个 Bean 实例，则可以编写为以下的方式：

```
<entry>
   <key>
      <ref bean="myKeyBean"/>
   </key>
   <ref bean="myValueBean"/>
</entry>
```

您还可以编写为以下的形式，更为简洁：

```
<entry key-ref="myKeyBean" value-ref="myValueBean"/>
```

如果您所使用的 JDK 是 5.0 以上的版本，在定义集合对象时，也许会使用泛型（Generic）的功能来定义集合对象所存放的类型，例如：

```
public class Some {
   private Map<String, Float> somes;

   public void setSomes(Map<String, Float> somes) {
      this.somes = somes;
   }
}
```

在进行依赖注入时，Spring 会尝试为属性作类型转换的动作，再将之放入集合对象中，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

例如：

```
...
   <bean id="some" class="onlyfun.caterpillar.Some">
      <property name="somes">
         <map>
            <entry key="Justin" value="99.99"/>
            <entry key="momor" value="99.75"/>
         </map>
      </property>
   </bean>
...
```

在这样的设置之下，Spring 会将 key 的值转换为 `String`，value 的设置转换为 `Float`，以符合 `Some` 类中 `somes` 属性的泛型声明。

如果要注入的是 *java.util.Set* 类型，则使用 `<set>` 标签，一个设置的片段如下所示：

```
...
<set>
   <value>a set element</value>
   <ref bean="otherBean"/>
   <ref bean="anotherBean"/>
</set>
...
```

若要注入的是 *java.util.Properties* 类型，则使用`<props>`与`<prop>`标签，Bean 定义文件的写法示范如下：

```
   <bean id=....>
      ....
      <property name="someProperties">
         <props>
            <prop key="someProkey1">someProValue1</prop>
            <prop key="someProkey2">someProValue2</prop>
         </props>
      </property>
   </bean>
```

如果您的集合对象不只注入一个对象，则要考虑为集合对象设置`"id"`名称，例如若是 `List` 对象，则要使用 `org.springframework.beans.factory.config.ListFactoryBean` 来定义，如下所示：

```
<bean id="emails"
     class="org.springframework.beans.factory.config.ListFactoryBean">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <property name="sourceList">
        <list>
            <value>justin@caterpillar.onlyfun</value>
            <value>momor@caterpillar.onlyfun</value>
        </list>
    </property>
</bean>
```

如果是 Map 对象，则可以如下设置：

```
<bean id="emails"
     class="org.springframework.beans.factory.config.MapFactoryBean">
    <property name="sourceMap">
        <map>
          <entry key="justin" value="justin@caterpillar.onlyfun"/>
          <entry key="momor" value="momor@caterpillar.onlyfun"/>
        </map>
    </property>
</bean>
```

如果是 Set 对象，则可以如下设置：

```
<bean id="emails"
     class="org.springframework.beans.factory.config.SetFactoryBean">
    <property name="sourceSet">
        <set>
            <value>justin@caterpillar.onlyfun</value>
            <value>momor@caterpillar.onlyfun</value>
        </set>
    </property>
</bean>
```

如果是 Properties 类型，则可以使用 org.springframework.beans.factory.config.PropertiesFactoryBean，例如：

```
<bean id="propConfiguration"
     class="org.springframework.beans.
              → factory.config.PropertiesFactoryBean">
    <props>
        <prop key="someProkey1">someProValue1</prop>
        <prop key="someProkey2">someProValue2</prop>
    </props>
</bean>
```

或是使用"location"属性，指定*.properties* 文件的位置，从中读取 Properties 资料，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<bean id="businessConfig"
     class="org.springframework.beans.
           → factory.config.PropertiesFactoryBean">
    <property name="location"
        value="classpath:onlyfun/caterpillar/config.properties"/>
</bean>
```

在 Spring 2.0 中，还支持集合对象的合并，这与 3.1.6 节谈到的 Bean 定义的继承类似，例如：

```
<bean id="parent" abstract="true" class="onlyfun.caterpillar.Some">
    <property name="someProperties">
        <props>
            <prop key="someProkey1">someProValue1</prop>
            <prop key="someProkey2">someProValue2</prop>
        </props>
    </property>
</bean>
<bean id="child" parent="parent">
    <property name="someProperties">
        <props merge="true">
            <prop key="someProkey2">otherProValue2</prop>
            <prop key="someProkey3">someProValue3</prop>
        </props>
    </property>
</bean>
```

注意："child"中的<props>将"merge"属性设置为"true"，以上的设置所取得的"child"，将会继承"parent"中的定义，重新定义"someProKey2"为"otherProValue2"，并新增"someProKey3"为"someProValue3"，也就是最后相当于定义了以下的*.properties* 文件：

```
someProKey1=someProValue1
someProKey2=otherProValue2
someProKey3=someProValue3
```

3.2.5 Spring 2.0 的<util>标签

在 Spring 2.0 中若使用基于 XML Schema 的 XML 定义文件进行设置，则可以加入新增的<util>标签扩充。<util>标签在设置 Bean 定义时更为方便省事，对于 XML 配置文件的简化有很大的帮助，要使用<util>标签，必须在 XML 中加入 util 名称空间（namespace）：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/util
  http://www.springframework.org/schema/util/spring-util-2.0.xsd">
    ...
</beans>
```

首先来介绍<util>标签在集合对象设置上的简化，可以分别使用<util:list>、<util:map>、<util:set>、<util:properties>等标签，用它来取代先前所介绍的ListFactoryBean、MapFactoryBean、SetFactoryBean 与 PropertiesFactoryBean。这里先以 CollectionDemo 项目为例，可以将其 XML 配置文件更换为以下的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/util
  http://www.springframework.org/schema/util/spring-util-2.0.xsd">

    <bean id="some1" class="onlyfun.caterpillar.Some">
        <property name="name" value="Justin"/>
    </bean>

    <bean id="some2" class="onlyfun.caterpillar.Some">
        <property name="name" value="momor"/>
    </bean>

    <util:list id="strArray">
        <value>Hello</value>
        <value>Welcome</value>
    </util:list>

    <util:list id="objArray">
        <ref bean="some1"/>
        <ref bean="some2"/>
    </util:list>

    <util:list id="list" list-class="java.util.ArrayList">
        <value>ListTest</value>
        <ref bean="some1"/>
        <ref bean="some2"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    </util:list>

    <util:map id="map" map-class="java.util.HashMap">
        <entry key="MapTest" value="Hello!Justin!"/>
        <entry key="someKey1" value-ref="some1"/>
    </util:map>

    <bean id="someBean" class="onlyfun.caterpillar.SomeBean">
        <property name="someStrArray" ref="strArray"/>
        <property name="someObjArray" ref="objArray"/>
        <property name="someList" ref="list"/>
        <property name="someMap" ref="map"/>
    </bean>

</beans>
```

其中"list-class"、"map-class"属性可以指定所要使用的集合对象，若不指定，则由容器自行决定实现的容器对象。以下是<util:properties>标签的使用范例：

```
<util:properties id="businessConfig"
    location="classpath:onlyfun/caterpillar/cpnfig.properties"/>
```

以下是<util:set>的使用范例，同样地，"set-class"可用来指定集合对象所使用的类型：

```
<util:set id="emails" set-class="java.util.TreeSet">
    <value>justin@caterpillar.onlyfun</value>
    <value>momor@caterpillar.onlyfun</value>
</util:set>
```

除了这里所介绍的<util>标签之外，还有<util:constant>可用来设置静态数据成员（static field），而免于设置 org.springframework.beans.factory.config.FieldRetrievingFactoryBean，例如：

```
<bean id="..." class="...">
  <property name="someProperty">
    <util:constant
      static-field="java.sql.Connection.TRANSACTION_SERIALIZABLE"/>
  </property>
</bean>
```

还可以使用<util:property-path>标签为某个 Bean 的属性成员设置"id"名称，使之在容器管理中，免于设置 org.springframework.beans.factory.config.PropertyPathFactoryBean，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<bean id="someBean" class="onlyfun.caterpillar.SomeBean">
  <property name="name" value="guest"/>
</bean>
<!--id 为 name 的 Bean，其值将会是 someBean.name-->
<util:property-path id="name" path="someBean.name"/>
```

3.2.6 Lookup Method Injection

假设现在要设计一个 Singleton 的 MessageManager，当调用 display()方法时，会取得一个系统新建立的 Message 对象并加以显示，例如：

LookupMethodInjectionDemo　　MessageManager.java

```
package onlyfun.caterpillar;

public abstract class MessageManager {
    public void display() {
        Message message = createMessage();
        System.out.println(message);
    }

    protected abstract Message createMessage();
}
```

Message 收集了系统当时的相关信息，这里的范例则只简单地收集了系统的时间信息：

LookupMethodInjectionDemo　　Message.java

```
package onlyfun.caterpillar;

import java.util.Date;

public class Message {
    private String sysMessage;

    public Message() {
        sysMessage = "系统资讯: " + new Date().toString();
    }

    public String toString() {
        return sysMessage;
    }
}
```

在这里不想编写程序直接实现 createMessage()方法，使之 new 一个 Message 对象并传回，而是希望通过 Spring 来产生并注入 Message 对象，以获得替换 Message 的弹性。在

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

XML 配置文件中单纯地将 Message 的"scope"属性设置为"prototype"是行不通的，因为只有在通过 BeanFactory 或 ApplicationContext 的 getBean()来取得 Message 时，才会重新实例化一个 Message。

这时可以使用 Spring 的 Lookup Method Injection，使用<lookup-method>标签，可指定使用某个 Bean 的方法，产生新的对象并进行注入，例如在 XML 配置文件中，可以这样编写：

LookupMethodInjectionDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="sysMessage"
         class="onlyfun.caterpillar.Message" scope="prototype"/>

    <bean id="messageManager"
         class="onlyfun.caterpillar.MessageManager">
       <lookup-method name="createMessage" bean="sysMessage"/>
    </bean>
</beans>
```

如此设置，Spring 将使用 CGLIB 产生一个 MessageManager 的子类实现（所以要将 CGLIB 的 *.jar* 加入 *Classpath* 之中），并且在每次调用 createMessage()方法时，建立一个 Message 对象并传回，可以使用一个简单的程序来示范执行结果：

LookupMethodInjectionDemo　　SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.
                  ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "beans-config.xml");
        MessageManager messageManager =
            (MessageManager) context.getBean("messageManager");

        try {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            messageManager.display();
            Thread.sleep(1000);
            messageManager.display();
            Thread.sleep(1000);
            messageManager.display();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

程序故意在每次调用 display()之时暂停 1000 毫秒，以区分系统的时间信息，观察 Message 对象是否确实重新产生。可以从执行的结果中看到，确实每次都产生新的 Message 对象，如图 3.6 所示。

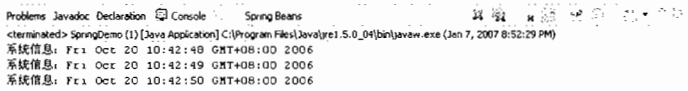


图 3.6 LookupMethodInjectionDemo 项目执行结果

Lookup Method Injection 是 Spring 所提供的 Method Injection 解决方案之一，从实际的例子中可以看到，它可以指定方法 createMessage()，由 Spring 提供实现，将指定的 Message 对象注入 MessageManager 之中。

Spring 中还提供一个更不常用的 Arbitrary method replacement，可以更进一步地取代某个对象的方法实现，且不用修改原来的对象。若有兴趣，可以直接参考 Spring 参考文件中 3.3.8.2. Arbitrary method replacement 的相关范例与说明。

3.3 Bean 高级管理

对于应用程序组件来说，最理想的情况是这些组件不需要意识到 Spring 容器的存在，具体来说就是不使用 Spring 的相关 API。然而您可以为应用程序组件编写一些 Spring 服务程序，让这些程序得知有关于 Spring 容器的一些消息，以获得 Spring 所提供的一些功能。另一方面，您可能必须让容器对应用程序组件进行一些额外处理，这节中将提供一些高级的 Bean 设置与管理的方式。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.3.1 非 XML 定义文件的配置方式

在先前所见到的每个 Spring 示范程序都使用 XML 作为定义文件，XML 文件的层次格式适用于配置设置，因此许多的开放源代码项目（Open source project）也都将 XML 作为预设的配置定义方式，但通常也会提供非 XML 定义文件的方式，例如 Spring 提供属性文件 *.properties* 的设置方式，但也可以使用属性文件 *.properties* 定义 Bean。

举个实际的例子来说，假设您定义了一个 HelloBean 类：

PropertiesDemo HelloBean.java

```
package onlyfun.caterpillar;

public class HelloBean {
    private String helloWord;

    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }

    public String getHelloWord() {
        return helloWord;
    }
}
```

在这里应编写一个 *beans-config.properties* 来定义 Bean 与依赖注入等内容：

PropertiesDemo beans-config.properties

```
helloBean.(class)=onlyfun.caterpillar.HelloBean
helloBean.helloWord=Welcome
```

属性文件中 "helloBean" 名称即 Bean 的别名设置，.(class)用于指定类来源，有括号表示 Bean 定义文件所需的属性设置，例如还有如.(abstract)、.(parent)、.(lazy-init)、.(singleton)等都可以设置，至于其他的属性设置方式，例如".helloWord"即为 Bean 的属性名称，如果要参考已存在的 Bean，则使用".helloWorld(ref)"。可以使用 org.springframework.beans.factory.support. PropertiesBeanDefinitionReader 来读取属性文件，读取 *.properties* 文件中的定义的示范程序，如下所示：

PropertiesDemo SpringDemo.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.
          support.BeanDefinitionRegistry;
import org.springframework.beans.factory.
          support.DefaultListableBeanFactory;
import org.springframework.beans.factory.
          support.PropertiesBeanDefinitionReader;
import org.springframework.core.io.ClassPathResource;

public class SpringDemo {
   public static void main(String[] args) {
      BeanDefinitionRegistry reg =
         new DefaultListableBeanFactory();
      PropertiesBeanDefinitionReader reader =
         new PropertiesBeanDefinitionReader(reg);
      reader.loadBeanDefinitions(
            new ClassPathResource(
                  "beans-config.properties"));

      BeanFactory factory = (BeanFactory) reg;
      HelloBean hello =
         (HelloBean) factory.getBean("helloBean");
      System.out.println(hello.getHelloWord());
   }
}
```

执行结果会显示 "Welcome" 文字。

除了通过 XML 或 *.properties* 文件，也可以在程序中直接编写程序，以定义 Bean 与 Bean 之间的依赖关系，通过 `org.springframework.beans.MutablePropertyValues` 设置属性，将属性与 Bean 的类设置给 `org.springframework.beans.factory.support.RootBeanDefinition`，并向 `org.springframework.beans.factory.support.BeanDefinitionRegistry` 注册。

不使用任何的文件来定义 Bean 的好处是，客户端与定义文件是隔离的，它们无法接触定义文件的内容，另一方面又可以使用 Spring 所提供的容器等相关功能。

直接来看个例子，可以将 PropertiesDemo 的 `SpringDemo` 类改写如下：

```
package onlyfun.caterpillar;

import org.springframework.beans.factory.
          support.BeanDefinitionRegistry;
import org.springframework.beans.factory.
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            support.DefaultListableBeanFactory;
import org.springframework.beans.factory.
            support.RootBeanDefinition;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.MutablePropertyValues;

public class SpringDemo {
    public static void main(String[] args) {
        // 设置属性
        MutablePropertyValues properties =
            new MutablePropertyValues();
        properties.addPropertyValue(
                "helloWord", "Hello!Justin!");

        // 设置 Bean 定义
        RootBeanDefinition definition =
            new RootBeanDefinition(
                    HelloBean.class, properties);

        // 注册 Bean 定义与 Bean 别名
        BeanDefinitionRegistry reg =
            new DefaultListableBeanFactory();
        reg.registerBeanDefinition("helloBean", definition);

        BeanFactory factory = (BeanFactory) reg;
        HelloBean hello =
            (HelloBean) factory.getBean("helloBean");
        System.out.println(hello.getHelloWord());
    }
}
```

执行结果会显示 "Hello!Justin!" 的文字。

3.3.2　Aware 相关接口

对于应用程序组件来说，最佳的情况是它根本不知道自己是被 Spring 容器所管理的，应用程序可以做自己该从事的工作或业务逻辑，而不会与 Spring 容器或相关 API 产生耦合关系。

然而有时候，为了善用 Spring 所提供的一些功能，必须让 Bean 知道 Spring 容器管理它的一些细节，例如必须让 Bean 知道自己在容器中是以哪个别名被管理的，或者让它知道 BeanFactory、ApplicationContext 的存在，也就是让 Bean 可以取得目前运行中的 BeanFactory 或 ApplicationContext 的实例。例如若 Bean 可以意识到 ApplicationContext 的存在，那么在 Bean 的某些动作发生时，可以进行事件发布，并让该事件感兴趣的接受者来处理。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Spring 中提供一些 Aware 相关接口，如 org.springframework.beans.factory.BeanNameAware、org.springframework.beans.factory.BeanFactoryAware、org.springframework.context.ApplicationContextAware 等，实现这些 Aware 接口的 Bean 类在被初始之后，可以取得一些 Spring 所提供的资源或使用某些功能。

例如实现 BeanNameAware 接口的 Bean 类，在设置依赖关系后、初始化方法之前（例如 InitializingBean 的 afterPropertiesSet() 方法或自定义的 init 方法），会将 Bean 在定义文件中的名称通过 setBeanName() 方法设置给 Bean，BeanNameAware 接口在 Spring 中的定义如下：

```
package org.springframework.beans.factory;
public interface BeanNameAware {
   public void setBeanName(String name);
}
```

实现 BeanFactoryAware 接口的 Bean 类，在设置依赖关系后、初始化方法之前，Spring 容器将会注入 BeanFactory 的实例，BeanFactoryAware 接口于 Spring 中的定义如下：

```
package org.springframework.beans.factory;
public interface BeanFactoryAware {
   public void setBeanFactory(BeanFactory beanFactory)
                                  throws BeansException;
}
```

而实现 ApplicationContextAware 接口的 Bean 类，在 Bean 类被初始后，将会被注入 ApplicationContext 的实例，ApplicationContextAware 接口于 Spring 中的定义如下：

```
package org.springframework.context;
public interface ApplicationContextAware {
   public void setApplicationContext(
      ApplicationContext context) throws BeansException;
}
```

另外也可以实现 org.springframework.context.ResourceLoaderAware 接口，可以让 Bean 取得 ResourceLoader 实例，并进一步取得相关的资源文件。

良葛格的话匣子<<<

当然了，一旦实现以上所提供的相关接口，应用程序组件就会因使用到 Spring 的相关 API 而与 Spring 产生耦合关系，所以像这样用来取得 Spring 资源的 Bean，设计上应该算是 Spring 框架服务的一部分，是应用程序组件与 Spring 框架的桥梁。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.3.3 BeanPostProcessor

在 Bean 的依赖关系由 Spring 容器建立并设置之后，您还有机会自定义一些 Bean 的修正动作来修正相关的属性，方法是让 Bean 类实现 org.springframework.beans.factory.config.BeanPostProcessor 接口，该接口于 Spring 中的定义如下：

```
package org.springframework.beans.factory.config;
public interface BeanPostProcessor {
    public Object postProcessBeforeInitialization(
           Object bean, String name) throws BeansException;
    public Object postProcessAfterInitialization(
           Object bean, String name) throws BeansException;
}
```

postProcessBeforeInitialization() 方法会在 Bean 类被初始化之前（例如 InitializingBean 的 afterPropertiesSet() 方法或自定义的 init 方法）被执行，而 postProcessAfterInitialization() 方法会在 Bean 类被初始化之后立即被执行。

举个例子来说，您可以实现一个英文字母转大写修正器类，作用是对于 String 类型的依赖注入，无论在定义文件中编写时是大写还是小写，在 Bean 的依赖注入设置之后，使用英文字母转大写修正器将所有注入的 String 都改为大写，英文字母转大写修正器 UppercaseModifier 类编写如下：

BeanPostProcessorDemo　　UppercaseModifier.java

```
package onlyfun.caterpillar;

import java.lang.reflect.Field;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.
          config.BeanPostProcessor;

public class UppercaseModifier
             implements BeanPostProcessor {

    public Object postProcessBeforeInitialization(
              Object bean, String name) throws BeansException {

        // 取得 Bean 的 Field 成员
        Field[] fields =
           bean.getClass().getDeclaredFields();
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        for(int i = 0; i < fields.length; i++) {
            // 针对 String 类型的 Field 成员加以修正
            if(fields[i].getType().equals(String.class)) {
                fields[i].setAccessible(true);
                try {
                    String original =
                        (String) fields[i].get(bean);
                    // 改为大写
                    fields[i].set(bean, original.toUpperCase());
                } catch (IllegalArgumentException e) {
                    e.printStackTrace();
                } catch (IllegalAccessException e) {
                    e.printStackTrace();
                }
            }
        }

        return bean;
    }

    public Object postProcessAfterInitialization(
                Object bean, String name) throws BeansException {
        return bean;
    }
}
```

配合这个类的设计，来示范一下如何应用，假设您定义了一个 HelloBean 类：

BeanPostProcessorDemo　　HelloBean.java

```
package onlyfun.caterpillar;

public class HelloBean {
    private String helloWord;

    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }

    public String getHelloWord() {
        return helloWord;
    }
}
```

在 Bean 定义文件的编写上，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

BeanPostProcessorDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="uppercaseModifier"
         class="onlyfun.caterpillar.UppercaseModifier"/>

    <bean id="helloBean" class="onlyfun.caterpillar.HelloBean">
       <property name="helloWord" value="Hello"/>
    </bean>

</beans>
```

注意到在"helloWord"属性上设置的 “Hello” 英文字母是有大写也有小写的，ApplicationContext 会自动侦测是否在定义文件中定义了实现 BeanPostProcessor 接口的类，Spring 容器会在每一个 Bean 被初始化之前、之后分别执行 uppercaseModifier 的 postProcessBeforeInitialization() 方法与 postProcessAfterInitialization() 方法，以对 Bean 进行指定的相关修正，可以实际来看看以下的测试程序：

BeanPostProcessorDemo SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
          support.ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "beans-config.xml");

        HelloBean hello =
            (HelloBean) context.getBean("helloBean");
        System.out.println(hello.getHelloWord());
    }
}
```

虽然在 Bean 定义文件中，"helloBean"的"helloWord"属性上设置的 “Hello” 英文字

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

母有大写也有小写，但执行结果中将会显示 “HELLO” 的文字，也就是所有的英文字母都被转换为大写字母。

3.3.4 BeanFactoryPostProcessor

在 BeanFactory 载入 Bean 定义文件的所有内容，但还没正式产生 Bean 实例之前，可以对该 BeanFactory 进行一些处理，方法是让 Bean 类实现 org.springframework.beans.factory.config.BeanFactoryPostProcessor 接口，该接口于 Spring 中的定义如下：

```
package org.springframework.beans.factory.config;
public interface BeanFactoryPostProcessor {
   public void postProcessBeanFactory(
         ConfigurableListableBeanFactory beanFactory)
                                    throws BeansException;
}
```

假设有一个 SomClass 实现 BeanFactoryPostProcessor 接口，则可以在 Bean 定义文件中定义它：

```
....
<beans>
   <bean id="someClass"
         class="onlyfun.caterpillar.SomeClass"/>
   ....
</beans>
....
```

使用 ApplicationContext 时，若在 Bean 定义文件中定义了实现 BeanFactoryPostProcessor 接口的类，则 ApplicationContext 会自动使用这些类的实例。

在 Spring 中提供有几个 BeanFactoryPostProcessor 接口的实现类，像 org.springframework.beans.factory.config.PropertyPlaceholderConfigurer、org.springframework.beans.factory.config.PropertyOverrideConfigurer 、 org.springframework.beans.factory.config.CustomEditorConfigurer，接下来会一个一个地为您介绍这几个类的运用方式。

3.3.5 PropertyPlaceholderConfigurer

Spring 提供了一个 BeanFactoryPostProcessor 接口的实现类：org.springframework.beans.factory.config.PropertyPlaceholderConfigurer。通过这个类，可以将一些配置设置信息移出至一个或多个 *.properties* 文件中，如此的安排可以让 XML 定义文件负责系统的相关设置，而 *.properties* 文件可以让客户根据实际应用时的需求，自定

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

义一些相关的参数。

来看一个 Bean 定义文件的实际例子，可以将之前的 BeanPostProcessorDemo 项目中的 *beans-config.xml* 修改为以下的内容：

PropertyPlaceholderConfigurerDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="configBean"
         class="org.springframework.beans.factory.
                → config.PropertyPlaceholderConfigurer">
       <property name="location" value="hello.properties"/>
    </bean>

    <bean id="helloBean" class="onlyfun.caterpillar.HelloBean">
       <property name="helloWord"
                value="${onlyfun.caterpillar.helloWord}"/>
    </bean>

</beans>
```

假设在"helloBean"中有许多依赖注入的属性，这些大都是不常变动的属性，而其中"helloWord"会经常变动，可以编写一个 *hello.properties* 文件来设置这个属性，而 *hello.properties* 文件的指定被设置在"configBean"的 "location"属性中，*hello.properties* 的内容编写如下：

PropertyPlaceholderConfigurerDemo hello.properties

```
onlyfun.caterpillar.helloWord=Welcome
```

在"helloBean"的"helloWord"属性中，${onlyfun.caterpillar.helloWord} 将会被 *hello.properties* 的 onlyfun.caterpillar.helloWord 的设置值取代，所以最后的结果会显示“Welcome”的文字。

如果有多个 *.properties* 文件，则可以通过 "locations" 属性来设置，例如：

```
...
<beans>
    <bean id="configBean"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    class="org.springframework.beans.factory.
          → config.PropertyPlaceholderConfigurer">
       <property name="locations">
          <list>
             <value>hello.properties</value>
             <value>welcome.properties</value>
             <value>other.properties</value>
          </list>
       </property>
    </bean>
    <bean id="helloBean"
          class="onlyfun.caterpillar.HelloBean">
       <property name="helloWord"
             value="${onlyfun.caterpillar.helloWord}"/>
    </bean>
</beans>
```

3.3.6 PropertyOverrideConfigurer

Spring 框架提供了一个 BeanFactoryPostProcessor 接口的实现类：org.springframework.beans.factory.config.PropertyOverrideConfigurer。通过这个类，您可以在 *.properties* 文件中设置一些优先的属性，当 *.properties* 文件中的某个设置与 XML 中的某个设置重复时，则 XML 中的设置会被覆盖，也就是以 *.properties* 文件中的设置值为主。

举个实际的程序作为示范，可以修改之前的 PropertyPlaceholderConfigurerDemo 项目中的 *beans-config.xml* 文件，如下所示。

PropertyOverrideConfigurerDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="configBean"
         class="org.springframework.beans.factory.
                → config.PropertyOverrideConfigurer">
       <property name="location" value="hello.properties"/>
    </bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean">
       <property name="helloWord" value="Hello"/>
    </bean>

</beans>
```

在 *hello.properties* 的编写上，假设内容如下：

PropertyOverrideConfigurerDemo　　hello.properties

```
helloBean.helloWord=Welcome
```

"helloBean"对应于 XML 定义文件中的某个 Bean 的 "id" 值，当中的"helloWord"设置将覆盖 XML 中的"helloWord"属性设置，如果执行先前编写的 SpringDemo 程序，则程序会显示 *.properties* 文件中设置的 “Welcome”文字，而不是 XML 文件中的 “Hello” 文字。

同样地，如果有多个 *.properties* 文件，则可以通过 "locations" 属性来设置，可以参考之前介绍 PropertyPlaceholderConfigurer 时的设置方式。

良葛格的话匣子<<<

基于篇幅的限制，这里并没有将修改过后完整的程序架构中所有的文件列出来，提醒一下，如果在范例标题上提供了项目名称，则可以在光盘中找到对应的完整范例项目，必要时可以观看一下完整的程序需要哪些文件以及如何编写。

3.3.7 CustomEditorConfigurer

Spring 框架提供了一个 BeanFactoryPostProcessor 接口的实现类：org.springframework.beans.factory.config.CustomEditorConfigurer。这个类可以读取实现 java.beans.PropertyEditor 接口的类，并按其中的实现，将字符串值转换为指定类型的对象。

举个例子来说，假设现在设计了两个类：User 类与 HelloBean 类。

CustomEditorConfigurerDemo　　User.java

```
package onlyfun.caterpillar;

public class User {
    private String name;
    private int number;

    public String getName() {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }
}
```

CustomEditorConfigurerDemo　　HelloBean.java

```
package onlyfun.caterpillar;

public class HelloBean {
    private String helloWord;
    private User user;

    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }
    public String getHelloWord() {
        return helloWord;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

若按之前介绍过的设置方式，基本上要在 Bean 定义文件中作以下的设置，以完成依赖注入：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="userBean" class="onlyfun.caterpillar.User">
        <property name="name" value="Justin"/>
        <property name="number" value="5858588"/>
    </bean>

    <bean id="helloBean" class="onlyfun.caterpillar.HelloBean">
        <property name="helloWord" value="Hello!"/>
        <property name="user" ref="userBean"/>
    </bean>
</beans>
```

事实上这边不采取以上的设置，可以在 Bean 定义文件中直接编写一个字符串设置，然后让 Spring 自动依指定的字符串转换为指定类型的对象，您可以实现 java.beans.PropertyEditor 接口，但更方便的是继承 java.beans.PropertyEditorSupport。

PropertyEditorSupport 实现了 PropertyEditor 接口，您可以重新定义它的 setAsText() 方法，这个方法传入一个字符串值，您可以根据这个字符串值内容来建立一个 User 对象，例如编写一个 UserEditor 类，它继承了 PropertyEditorSupport 类：

CustomEditorConfigurerDemo — UserEditor.java

```
package onlyfun.caterpillar;

import java.beans.PropertyEditorSupport;

public class UserEditor extends PropertyEditorSupport {
    public void setAsText(String text) {
        String[] strs = text.split(",");
        int number = Integer.parseInt(strs[1]);

        User user = new User();
        user.setName(strs[0]);
        user.setNumber(number);

        setValue(user);
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

UserEditor 类会按传入的字符串以 "," 分隔设置，将之分为两个部分的信息，之后产生一个 User 实例，再将这两个信息设置给 User 实例上的对应属性，为了运用这个 UserEditor 类，您要在 Bean 定义文件中使用 CustomEditorConfigurer 类，并设置它的 "customEditors" 属性，例如：

CustomEditorConfigurerDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="configBean"
         class="org.springframework.beans.factory.
                  → config.CustomEditorConfigurer">
       <property name="customEditors">
          <map>
             <entry key="onlyfun.caterpillar.User">
                <bean id="userEditor"
                      class="onlyfun.caterpillar.UserEditor"/>
             </entry>
          </map>
       </property>
    </bean>

    <bean id="helloBean"
         class="onlyfun.caterpillar.HelloBean">
       <property name="helloWord" value="Hello"/>
       <property name="user" value="Justin,5858588"/>
    </bean>

</beans>
```

只要是 User 类型的属性，都可以使用 "Justin,5858588" 这样的字符串来设置，CustomEditorConfigurer 会载入 "customEditors" 属性中设置的 key-value，根据 key 遇到哪一个类型时要使用 PropertyEditor，value 设置就指定对应的 PropertyEditor。

为了验证设置是否正确，来简单地编写一个测试程序执行，看看结果如何：

CustomEditorConfigurerDemo　　SpringDemo.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.
          support.ClassPathXmlApplicationContext;

public class SpringDemo {
   public static void main(String[] args) {
      ApplicationContext context =
         new ClassPathXmlApplicationContext(
               "beans-config.xml");

      HelloBean hello =
         (HelloBean) context.getBean("helloBean");
      System.out.println(hello.getHelloWord() + "!");
      System.out.println("Name:\t"
            + hello.getUser().getName());
      System.out.println("Number:\t"
            + hello.getUser().getNumber());
   }
}
```

执行结果如图 3.7 所示，您可以通过 User 的实例直接取得 Bean 定义文件中设置的值：

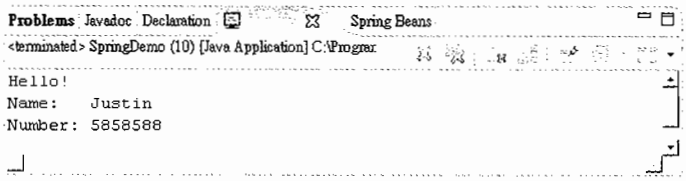


图 3.7　CustomEditorConfigurerDemo 项目执行结果

3.4　资源、消息、事件

ApplicationContext 除了具备如 BeanFactory 基本的容器管理功能之外，还支持更多应用程序框架的特性，像资源的取得、消息解析、事件的处理与传播，您可以基于 Spring 容器来打造属于自己的应用程序或框架。

3.4.1　资源的取得

Spring 提供了对资源文件的泛型存取（Generic access），ApplicationContext 继承了 org.springframework.core.io.ResourceLoader 接口，您可以使用 get-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Resource() 方法并指定资源文件的 URL（Uniform Resource Locator）来取得一个实现 Resource 接口的实例，例如：

```
Resource resource = context.getResource(
                    "classpath:admin.properties");
```

"classpath:" 是 Spring 自制的 URL 虚拟协定，这会取回一个 org.springframework.core.io.ClassPathResource 的实例，代表一个具体的资源文件，在上面的程序片段中，该资源文件是位于 *Classpath* 根目录中的，文件名称为 *admin.properties*，您也可以指定标准的 URL，像"file:" 或 "http:"，例如：

```
Resource resource = context.getResource(
  "file:c:/workspace/springtest/conf/admin.properties");
```

这会回传一个 org.springframework.core.io.FileSystemResource 的实例，或者您可以按如下指定来回传一个 ServletContextResource 实例：

```
Resource resource = context.getResource("WEB-INF/conf/admin.properties");
```

取得一个实现 Resource 接口的实例之后，您可以使用 getFile()、getInputStream() 等方式来操作或取得资源文件的相关资源，Resource 接口的实例只是资源文件的一个抽象代表，指定的资源文件实际上可能并不存在，您可以使用 exists() 方法来进行测试。

3.4.2 解析文字消息

ApplicationContext 继承了 org.springframework.context.MessageSource 接口，您可以使用 getMessage() 的各种版本的方法来取得文字消息的资源文件，从而实现国际化消息的目的。

您可以简单地通过 MessageSource 的一个实现 org.springframework.context.support.ResourceBundleMessageSource 来取得国际化消息，直接以实例进行说明，首先在 Bean 定义文件中编写以下的内容：

MessageSourceDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="messageSource"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        class="org.springframework.context.
                → support.ResourceBundleMessageSource">
      <property name="basename" value="messages"/>
   </bean>
</beans>
```

"basename" 属性用来设置消息资源的前置档案文件名称，在这边设置是 "messages"，表示消息资源文件的名称可以是 *messages_en_US.properties*、*messages_zh_TW.properties*、*messages_*.properties* 等名称，在这里提供两个：*messages_en_US.properties*、*messages_zh_TW.properties*，首先来看看 *messages_en_US.properties* 的内容编写，如下所示。

MessageSourceDemo messages_en_US.properties

```
userLogin=User {0} login at {1}
```

这是美式英文语系的文字消息文件，若使用美式英文语系设置时，将取得这个文字消息文件的内容并显示出来，其中 {0} 与 {1} 则是为了可以在程序执行过程中，替换实际文字而设置的占位符。

另外再来提供 *messages_zh_TW.properties* 文件，由于直接使用 Big5 中文编码编写的文字文件，在读取时会因编码问题出现乱码，所以您必须先编写一个 *messages_zh_TW.txt* 文件，内容如下：

MessageSourceDemo messages_zh_TW.txt

```
userLogin=用户 {0} 于 {1} 登录
```

然后使用 JDK 工具程序中的 native2ascii 进行编码转换，执行指令的方式如下：

```
native2ascii messages_zh_TW.txt messages_zh_TW.properties
```

转换之后的 *messages_zh_TW.properties* 文件内容是经过编码转换后的 Unicode 编码，内容如下所示：

MessageSourceDemo messages_zh_TW.properties

```
userLogin=\u4f7f\u7528\u8005 {0} \u65bc {1} \u767b\u5165
```

ResourceBundleMessageSource 实际上会使用 JDK 的 ResourceBundle 来读取这些文字消息文件，接下来看看如何进行语系相关设置，并看看在不同的语系之下所读取的各是哪个文字消息文件：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

MessageSourceDemo SpringDemo.java

```
package onlyfun.caterpillar;

import java.util.Calendar;
import java.util.Locale;
import org.springframework.context.ApplicationContext;
import org.springframework.context.
            support.ClassPathXmlApplicationContext;

public class SpringDemo {
   public static void main(String[] args) {
      ApplicationContext context =
         new ClassPathXmlApplicationContext(
               "beans-config.xml");

      Object[] arguments =
         new Object[] {"良葛格",
                     Calendar.getInstance().getTime()};

      System.out.println(
            context.getMessage("userLogin",
                           arguments,
                           Locale.US));
      System.out.println(
            context.getMessage("userLogin",
                            arguments,
                            Locale.TAIWAN));
   }
}
```

由于项目中 *beans-config.xml* 正好在 *Classpath* 中，所以程序中也可使用 ClassPathXmlApplicationContext 来读取，文字消息文件的选择根据是按一个 java.util.Locale 对象来决定的，在使用 getMessage() 方法取得消息时可以指定一个 Locale 对象，范例程序中分别指定了 Locale.US 代表 en_US 语系，Locale.TAIWAN 代表 zh_TW 语系，实际执行程序的结果如图 3.8 所示。

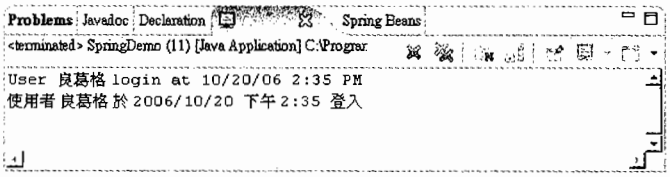


图 3.8 MessageSourceDemo 项目执行结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3.4.3 监听事件

在 Spring 应用程序执行期间，ApplicationContext 本身就会发布一连串的事件，所有发布的事件都是抽象类 org.springframework.context.ApplicationEvent 的子类，例如：

- ContextClosedEvent

 在 ApplicationContext 关闭时发布事件。

- ContextRefreshedEvent

 在 ApplicationContext 初始或 Refresh 时发布事件。

- RequestHandledEvent

 在 Web 应用程序中，当请求被处理时，ApplicationContext 会发布此事件。

 如果您对这些事件有兴趣，则可以实现 org.springframework.context.ApplicationListener 接口，并在定义文件中定义实现该接口的一个 Bean 实例：

```
package org.springframework.context;
import java.util.EventListener;

public interface ApplicationListener extends EventListener {
    void onApplicationEvent(ApplicationEvent event);
}
```

 ApplicationContext 会在 ApplicationEvent 发布时通知实现了 ApplicationListener 的 Bean 类实例。

 在 Spring 提供的 API 中，实现 ApplicationListener 接口的类有：

- ConsoleListener

 仅在 Debug 时使用，会在文字模式下显示记录（Log）ApplicationContext 的相关消息。

- PerformanceMonitorListener

 在 Web 应用程序中，搭配 WebApplicationContext 使用，可记录请求的回应时间。

 例如，可以在 Bean 定义文件中这么编写：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<beans ...>
    <bean id="listener"
          class="org.springframework.context.
                  → event.ConsoleListener"/>
    ....
</beans>
```

当运行 Spring 应用程序时，ApplicationEvent 相关事件会发布，您可以在文字模式下观察到相关的登录消息。

3.4.4 事件传播

如果打算发布事件通知 ApplicationListener 的实例（例如在应用程序中定时发布“心跳”事件，让应用程序的客户端可以知道应用程序是否还活着），可以使用 ApplicationContext 的 publishEvent() 方法，例如：

```
ApplicationContext context =
          new ClassPathXmlApplicationContext(
                    "beans-config.xml");
....
context.publishEvent(new ContextClosedEvent(context));
```

假设您在 Bean 定义文件中有这么定义：

```
...
<beans ...>
    <bean id="listener"
          class="org.springframework.context.
                  → event.ConsoleListener"/>
    ....
</beans>
```

则在文字模式下会显示事件发生时 ApplicationContext 的相关消息：

```
...
org.springframework.context.event.ContextClosedEvent[source=org.springframewo
rk.context.support.ClassPathXmlApplicationContext: display name
[org.springframework.context.support.ClassPathXmlApplicationContext;hashCode=
8795033]; startup date [Sat Oct 28 22:24:15 GMT+08:00 2006]; root of context
hierarchy]
```

接下来示范一个简单的自定义 ApplicationListener 与 ApplicationEvent 的例子，首先假设您继承了 ApplicationEvent 类，自行定义一个 SomeEvent 类，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

PublishEventDemo SomeEvent.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationEvent;

public class SomeEvent extends ApplicationEvent {
    public SomeEvent(Object source) {
        super(source);
    }
}
```

在构造 SomeEvent 时，您传递一个事件来源对象给它，接着您希望自定义一个 CustomListener，在 SomeEvent 被发布时，可以接收到并处理该事件，CustomListener 类的定义如下所示：

PublishEventDemo CustomListener.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationEvent;
import org.springframework.context.ApplicationListener;

public class CustomListener implements ApplicationListener {
    public void onApplicationEvent(ApplicationEvent event) {
        if(event instanceof SomeEvent) {
            System.out.println(event.getSource());
        }
    }
}
```

CustomListener 类实现了 ApplicationListener 接口，在 onAppli cationEvent() 方法中，先判断事件是不是 SomeEvent，如果是，则将事件来源显示在文字模式下。

在 Bean 定义文件的编写中，只要简单地定义一下 CustomListner 即可，例如：

PublishEventDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="listener"
          class="onlyfun.caterpillar.CustomListener"/>
</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在这里再编写一个简单的示范程序，直接发布一个 SomeEvent 事件，看看 CustomListener 是不是可以接收到该事件：

PublishEventDemo　　SpringDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
            support.ClassPathXmlApplicationContext;

public class SpringDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                    "beans-config.xml");
        context.publishEvent(new SomeEvent(SpringDemo.class));
    }
}
```

执行的结果如图 3.9 所示。

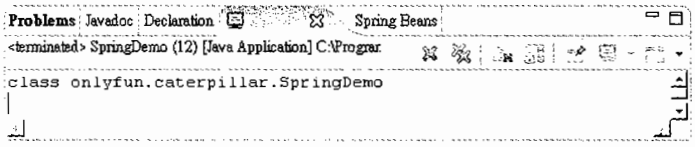


图 3.9　PublishEventemo 项目执行结果

3.5　接下来的主题

认识 Spring 核心容器是使用 Spring 的第一步，除了获得容器管理功能之外，在设置上一致的方式，使得 Spring 在使用上相当地简易。Spring 的另一个核心特色，也是所有子框架的基础，就是 Spring 的 AOP 支持。这是 Spring 的基础，也是 Spring 另一个特色，在下一个章节中，将带领您一步一步认识 AOP，以及如何使用 Spring 来实现 AOP。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 3.10　轻量级的 Spring 容器

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第 Spring AOP 章

在一个服务的流程中插入与业务逻辑无关的系统服务逻辑（例如 Logging、Security），这样的逻辑称为 Cross-cutting concerns，将 Crossing-cutting concerns 独立出来设计为一个对象，这样的特殊对象称之为 Aspect，Aspect-oriented programming 着重在 Aspect 的设计上以及与应用程序的织入（Weave）。以上各式各样与 AOP 相关的名词，在这个章节中都将获得解释。

AOP 跟 OOP（Object-oriented programming）并不相互抵触，它们是可以相辅相成的两个设计模型，Spring AOP 是实现 AOP 的一种技术，而 Spring AOP 也是 Spring 中一些子框架或子功能所依赖的核心，让您了解如何使用 Spring AOP 也是这个章节的重点之一。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.1 AOP 入门

AOP 全名 Aspect-oriented programming，对于尚未接触过 AOP 的人来说，AOP 中许多抽象的术语名词令人难以理解，像 Cross-cutting concerns、Aspect、Weave 等。在这节中，将先从代理机制（Spring 实现 AOP 的一种方式）来看一个实际且简单的示范程序，从而介绍 AOP 的观念与各种术语。

4.1.1 从代理机制初探 AOP

暂且先将 AOP 这个英文缩写名词放到一边，先从一个简单常见的例子来看一个议题，这个例子当中含有日志（Logging）动作，程序中常需要为某些动作或事件作下记录，以便在事后检查程序运作过程，或是作为出错时的信息。

来看一个最简单的例子。当您需要在执行某些方法时留下日志消息，凭直觉，您可能会如下编写：

```
package onlyfun.caterpillar;

import java.util.logging.*;

public class HelloSpeaker {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    public void hello(String name) {
        // 方法执行开始时留下记录
        logger.log(Level.INFO, "hello method starts....");
        // 程序主要功能
        System.out.println("Hello, " + name);
        // 方法执行完毕前留下记录
        logger.log(Level.INFO, "hello method ends....");
    }
}
```

在 HelloSpeaker 类中，当执行 hello()方法时，您希望方法在开始执行与执行完毕时都能留下记录，最简单的作法就是如以上的程序设计，在方法执行的前后加上日志动作，然而日志的这几行程序代码横切入（Cross- cutting）HelloSpeaker 类中，对于 HelloSpeaker 来说，日志的这几个动作并不属于 HelloSpeaker 业务逻辑（显示“Hello”等文字），这使得 HelloSpeaker 增加了额外的职责。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

想想如果程序中这种日志的动作到处都有需求，以上的写法势必造成您到处编写这些日志动作的程序代码，这使得维护日志程序代码的难度加大。如果需要的服务（Service）不只是日志动作，有一些非对象本身职责的相关动作也混入了对象之中（如权限检查、事务管理等），这会使得对象的负担进一步加重，甚至混淆了对象本身该负有的职责，对象本身的职责所占的程序代码，或许还少于这些与对象职责不相关的动作或服务的程序代码。

另一方面，使用以上的写法，若不再需要日志（或权限检查、事务管理等）的服务，那么将需要修改所有留下日志动作的程序代码，您无法简单地将这些相关服务从既有的程序中移去。

可以使用代理（Proxy）机制来解决这个问题，在这里讨论两种代理方式：静态代理（Static proxy）与动态代理（Dynamic proxy）。

- **静态代理**

在静态代理的实现中，代理对象与被代理对象必须实现同一个接口，在代理对象中可以实现日志等相关服务，并在需要的时候再调用被代理的对象，如此，被代理对象当中就可以仅保留与业务相关的职责。

举个实际的例子来说，首先定义一个 IHello 接口：

StaticProxyDemo　　IHello.java

```
package onlyfun.caterpillar;

public interface IHello {
    public void hello(String name);
}
```

然后让实现业务逻辑的 HelloSpeaker 类实现 IHello 接口，例如：

StaticProxyDemo　　HelloSpeaker.java

```
package onlyfun.caterpillar;

public class HelloSpeaker implements IHello {
    public void hello(String name) {
        System.out.println("Hello, " + name);
    }
}
```

可以看到，在 HelloSpeaker 类中现在没有任何日志的程序代码插入其中，日志服务的实现将被放至代理对象之中，代理对象同样也要实现 IHello 接口，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

StaticProxyDemo　　HelloProxy.java

```
package onlyfun.caterpillar;

import java.util.logging.*;

public class HelloProxy implements IHello {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    private IHello helloObject;

    public HelloProxy(IHello helloObject) {
        this.helloObject = helloObject;
    }

    public void hello(String name) {
        // 日志服务
        log("hello method starts....");

        // 执行业务逻辑
        helloObject.hello(name);

        // 日志服务
        log("hello method ends....");
    }

    private void log(String msg) {
        logger.log(Level.INFO, msg);
    }
}
```

在 HelloProxy 类的 hello() 方法中，要真正实现业务逻辑前后可以安排日志服务，实际编写一个测试程序来看看如何使用代理对象。

StaticProxyDemo　　ProxyDemo.java

```
package onlyfun.caterpillar;

public class ProxyDemo {
    public static void main(String[] args) {
        IHello proxy =
            new HelloProxy(new HelloSpeaker());
        proxy.hello("Justin");
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序中调用执行的是代理对象，构造代理对象时必须给它一个被代理对象，记得在操作取回代理对象时，必须转换操作接口为 IHello 接口，来看一下实际执行的结果，如图 4.1 所示。



图 4.1 StaticProxyDemo 项目执行结果

代理对象 HelloProxy 将代替真正的 HelloSpeaker 来执行 hello()，并在其前后加上日志的动作，这使得 HelloSpeaker 在编写时不必介入日志动作，HelloSpeaker 可以专心于它的职责，可以从图解的方式来更进一步看出代理机制的运作流程，如图 4.2 所示。

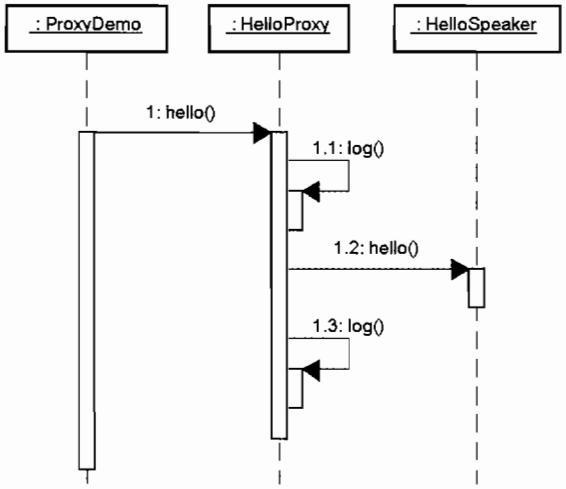


图 4.2 代理机制的运作流程

这是静态代理的基本范例，然而如您所看到的，代理对象的一个接口只服务于一种类型的对象，如果要代理的方法很多，势必要为每种方法进行代理，静态代理在程序规模稍大时就无法胜任了，在这里介绍静态代理的目的，是让您了解代理的基本原理。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

良葛格的话匣子<<<

很多人会问：图 4.1 的日志消息为何都出现在“Hello, Justin”文字之前（您执行的结果也许是在之后）？范例执行结果有错吗？要知道，日志的动作是由另一个线程来进行的，日志动作并不影响（或介入）程序流程，范例的重点在于日志动作确实有执行，而不是在于消息在主控台出现的顺序，日志动作也不是用来预测程序执行顺序之用。如果真的想看到主控台出现消息的顺序是否符合程序中对象与方法的调用顺序，那么将日志的部分全部改成标准输出即可。

- **动态代理**

在 JDK 1.3 之后加入了可协助开发动态代理功能的 API，您不必为特定的对象与方法编写特定的代理对象。使用动态代理，可以使得一个处理者（Handler）服务于各个对象。首先，一个处理者的类设计必须实现 `java.lang.reflect.InvocationHandler` 接口，以实例来进行说明，例如设计一个 `LogHandler` 类：

DynamicProxyDemo　　LogHandler.java

```
package onlyfun.caterpillar;

import java.util.logging.*;
import java.lang.reflect.*;

public class LogHandler implements InvocationHandler {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    private Object delegate;

    public Object bind(Object delegate) {
        this.delegate = delegate;
        return Proxy.newProxyInstance(
                        delegate.getClass().getClassLoader(),
                        delegate.getClass().getInterfaces(),
                        this);
    }

    public Object invoke(Object proxy, Method method,
                         Object[] args) throws Throwable {
        Object result = null;

        try {
            log("method starts..." + method);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            result = method.invoke(delegate, args);

            logger.log(Level.INFO, "method ends..." + method);
        } catch (Exception e){
            log(e.toString());
        }

        return result;
    }

    private void log(String message) {
        logger.log(Level.INFO, message);
    }
}
```

主要的概念是使用 Proxy.newProxyInstance() 静态方法建立一个代理对象，建立代理对象时必须告知所要代理的接口，之后您可以操作所建立的代理对象。在每次操作时会执行 InvocationHandler 的 invoke() 方法，invoke() 方法会传入被代理对象的方法名称与执行参数，实际上要执行的方法会交由 method.invoke()，您在 method.invoke() 前后加上日志动作，method.invoke() 传回的对象是实际方法执行过后的回传结果。

要实现动态代理，同样必须定义所要代理的接口，例如：

DynamicProxyDemo — IHello.java

```
package onlyfun.caterpillar;

public interface IHello {
    public void hello(String name);
}
```

然后让实现业务逻辑的 HelloSpeaker 类实现 IHello 接口，例如：

DynamicProxyDemo — HelloSpeaker.java

```
package onlyfun.caterpillar;

public class HelloSpeaker implements IHello {
    public void hello(String name) {
        System.out.println("Hello, " + name);
    }
}
```

眼尖的您或许发现了，这跟之前 StaticProxyDemo 项目中的 IHello 接口、HelloSpeaker 是相同的内容，在这里编写出来是为了范例的完整呈现。接下来编写一个测试的程序，您要

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用 LogHandler 的 bind() 方法来绑定被代理对象，如下所示：

DynamicProxyDemo　　ProxyDemo.java

```
package onlyfun.caterpillar;

public class ProxyDemo {
    public static void main(String[] args) {
        LogHandler logHandler  = new LogHandler();

        IHello helloProxy =
               (IHello) logHandler.bind(new HelloSpeaker());
        helloProxy.hello("Justin");
    }
}
```

来看一下执行的结果，如图 4.3 所示。

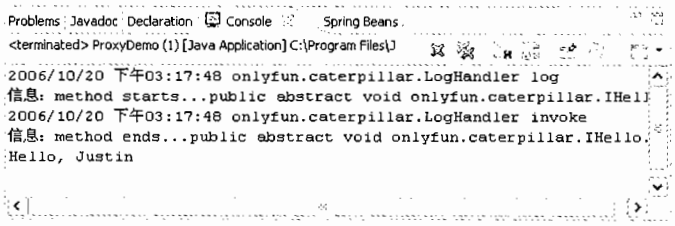


图 4.3　DynamicProxyDemo 项目的执行结果

LogHandler 不再服务于特定对象与接口，而 HelloSpeaker 也不用插入任何有关于日志的动作，它不用意识到日志动作的存在。

回到 AOP 的议题上，这个例子与 AOP 有何关系？

如 4.2 节中的例子，HelloSpeaker 本身的职责是显示招呼文字，却必须插入日志动作，这使得 HelloSpeaker 的职责加重，用 AOP 的术语来说，日志的程序代码横切（Cross-cutting）入 HelloSpeaker 的程序执行流程中，日志这样的动作在 AOP 中称之为横切关切点（Cross-cutting concern）。

使用代理对象将日志等与业务逻辑无关的动作或任务提取出来，设计成一个服务对象，像之前范例中示范的 HelloProxy 或是 LogHandler，这样的对象称之为切面（Aspect）。

AOP 中的 Aspect 所指的像日志等这类的动作或服务，将这些动作（Cross-cutting concerns）设计为通用、不介入特定业务对象的一个职责清楚的 Aspect 对象，这就是所谓的 Aspect-oriented programming，缩写名词即为 AOP。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在好的设计之下，Aspect 可以独立于应用程序之外，在必要的时候，可以介入应用程序之中提供服务，在不需要相关服务的时候，又可以将这些 Aspect 直接从应用程序中脱离出来，而应用程序本身不需修改任何一行程序代码。

良葛格的话匣子<<<

AOP 的相关英文名词术语要翻成中文实在很拗口，而且这些英文名词术语现在也没有统一的中文翻译名词，因此接下来的章节中，将以 AOP 原本的英文名词术语来呈现，将中文翻译的名词问题，留待日后更令人信服的中文名词。

4.1.2 AOP 观念与术语

AOP 全名为 Aspect-Oriented Programming，有关于 AOP 的许多名词术语都过于抽象，单从字面上并不容易理解其名词意义，这里将根据之前介绍代理机制的范例，来逐一对照以介绍 AOP 的术语与观念：

- Cross-cutting concern

在 DynamicProxyDemo 项目的例子中，日志的动作原先被横切（Cross-cutting）入至 `HelloSpeaker` 本身所负责的业务流程之中，另外类似于日志这类的动作，如安全（Security）检查、事务（Transaction）等系统层面的服务（Service），在一些应用程序之中常被见到安插至各个对象的处理流程之中，这些动作在 AOP 的术语中称为 Cross-cutting concerns。

以图片说明可强调出 Cross-cutting concerns 的概念，例如原来的业务流程是很单纯的，如图 4.4 所示。

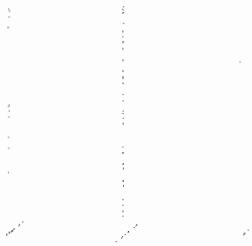


图 4.4 AOP 图解之一

现在为了要加入日志与安全检查等服务，对象的程序代码中被硬生生地写入相关的日志、安全检查程序片段，可使用图 4.5 表示出 Crosscutting 与 Cross-cutting concerns 的概念：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

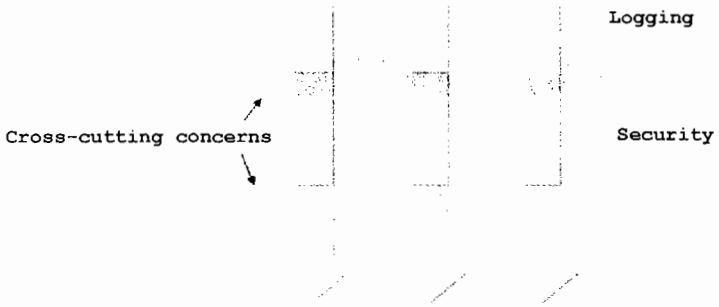


图 4.5　AOP 图解之二

Cross-cutting concerns 若直接编写在负责某业务的对象流程中，会使得维护程序的成本增高，若今天您要修改或移除对象中的日志功能，则必须修改所有编写日志服务的程序代码，然后重新编译，另一方面，Cross-cutting concerns 混杂于业务逻辑之中，使得业务对象本身的逻辑或程序的编写更为复杂。

- Aspect

将散落于各个业务逻辑之中的 Cross-cutting concerns 收集起来，设计成各个独立可重用的对象，这些对象称为 Aspect。例如在 DynamicProxyDemo 项目中，将日志的动作设计为一个 `LogHandler` 类，`LogHandler` 类在 AOP 的术语就是 Aspect 的一个具体实例。在 AOP 中着重于 Aspect 的辨认，使之从业务流程中独立出来。在需要该服务的时候，织入（Weave）至应用程序之上；不需要服务的时候，也可以马上从应用程序中脱离，且应用程序中的可重用组件不用作任何的修改。例如在 DynamicProxyDemo 项目中的 `HelloSpeaker` 所代表的角色就是应用程序中可重用的组件，在它需要日志服务时并不用修改本身的程序代码。

另一方面，对于应用程序中可重用的组件来说，按照 AOP 的设计方式，它不用知道提供服务的对象是否存在，具体地说，与服务相关的 API 不会出现在可重用的应用程序组件之中，因而可提高这些组件的重用性，您可以将这些组件应用至其他的应用程序之中，不会因为加入了某些服务而与目前的应用程序框架发生耦合。

- Advice

Aspect 当中对 Cross-cutting concerns 的具体实现称之为 Advice。以日志的动作而言，Advice 中会包括日志程序代码是如何实现的，像 DynamicProxyDemo 项目中 `LogHandler` 的 `invoke()`方法，就是 Advice 的一个具体实例。Advice 中包括了 Cross-cutting concerns 的行为或所要提供的服务。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- Joinpoint

Advice 在应用程序执行时加入业务流程的点或时机称之为 Joinpoint，具体来说，就是 Advice 在应用程序中被执行的时机。Spring 只支持方法的 Jointpoint，执行时机可能是某个方法被执行之前或之后（或两者都有），或是方法中某个异常发生的时候。

- Pointcut

Pointcut 定义了感兴趣的 Jointpoint，当调用的方法符合 Pointcut 表示式时，将 Advice 织入至应用程序上提供服务。具体地说在 Spring 2.0 中，您可以在定义文件或 Annotation 中编写 Pointcut，说明哪些 Advice 要应用至方法的前后。

- Target

一个 Advice 被应用的对象或目标对象，例如 DynamicProxyDemo 项目中的 `HelloSpeaker` 就是 `LogHandler` 中 Advice 的 Target。

- Introduction

对于一个现存的类，Introduction 可以为其增加行为，且不用修改该类的程序，具体来说，可以为某个已编写或编译完的类，在执行时期动态地加入一些方法或行为，而不用修改或新增任何一行程序代码。

- Proxy

在《Expert One-on-One J2EE Development Without EJB》一书中，Rod Johnson、Juergen Hoeller 在第 8 章中提到，AOP 的实现有 5 个主要的策略：Dynamic Proxies、Dynamic Byte Code Generation、Java Code Generation、Use of a Custon Class Loader、Language Extensions。

在之前的静态代理与动态代理中，已经使用了实际的程序范例介绍过的代理机制，Spring 的 AOP 主要是通过动态代理来完成的，可用于代理任何的接口。另一方面，Spring 也可以使用 CGLIB 代理，用以代理类，像一些遗留类（Legacy classes）。

- Weave

Advice 被应用至对象之上的过程称之为织入（Weave），在 AOP 中织入的方式有几个时间点：编译时期（Compile time）、类加载时期（Classload time）、执行时期（Runtime）。

结合 DynamicProxyDemo 的实例来说，将以上介绍过的与 AOP 相关的名词使用具体的图片来表示，有助于对每一个名词的理解与认识，如图 4.6 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

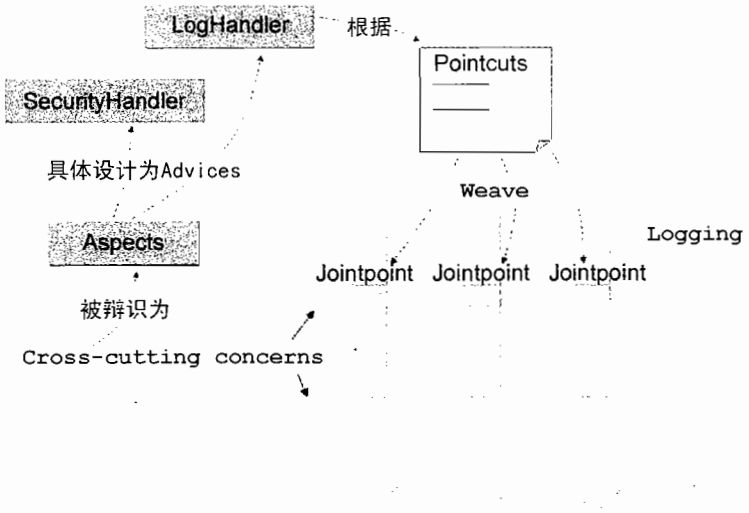


图 4.6　AOP 的各个名词示意

4.1.3　Spring AOP

不同的 AOP 框架会有其对 AOP 概念的不同实现方式，主要的差别在于所提供的 Pointcuts、Aspects 的丰富程度，以及它们如何被织入（Weave）至应用程序（像 Pointcuts 的定义方式）、代理的方式等。

Spring 的 Advices 是用 Java 程序语言来编写的，它不使用特定的 AOP 语言，在定义 Pointcuts 时可以使用 XML 配置文件或 Annotation，编写的方式对于 Java 开发人员来说都很熟悉，您不必学习特定的语法，就可以用熟悉的 Java 程序语言与 XML 格式来运用 Spring AOP。

若选择以实现 API 的方式来实现 Advices，Spring 的 AOP 会实现 AOP Alliance（*http://www.sourceforge.net/ projects/aopalliance*）所规范的接口，AOP Alliance 是由许多团体组成的联合项目（Joint project），这些团体对于 AOP 的实现要求是必须遵循制订出来的接口规范，目的是对 Java 的 AOP 实现接口标准化，以增加 AOP 实现类在不同的 Java 应用程序之间的可移植性。

在 Spring 2.0 中，您的 Advices 可以不实现任何接口，直接在设置文件中定义相关信息，就为 Targets 导入 Advices。您也可以选择使用 Annotation 的方式，在类之中直接标示，由 Spring 为您完成所有的代理对象设置。

Spring 的 Advices 是在执行时期导入至 Targets 的，您可以让 Targets 实现预先定义好的接口，Spring 在执行时期会使用 `java.lang.reflect.Proxy` 来进行动态代理，如果不实现

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接口，Spring 则会使用 CGLIB 为 Targets 产生一个子类作为代理类（Proxy classes）。

在 Spring AOP 中，您应该优先采用实现接口的方式，这可以让应用程序组件彼此之间的耦合度降低。若使用 Proxy classes 的方式，由于必须产生子类，所以对于被声明为 `final` 的方法无法进行代理，而且这个方式基本上是让一些无法更新源代码的第三方（Third-party）类或是旧类（Legacy classes）来使用的。

Spring 只支持方法的 Joinpoints，也就是 Advices 将在方法执行的前后被应用，Spring 不支持 Field 成员的 Jointpoints，这是因为在 Spring 的设计哲学中认为，支持 Field 成员的 Joinpoints 会破坏对象的封装性。

在 Spring 2.0 中提供了 3 种实现 AOP 的方式。

第一种是实现 Spring API 的传统方式，也就是用 Spring 1.2 或先前的版本来实现 AOP 的方式。您必须实现 Spring 当中的 AOP API 来定义 Advice，并设置代理对象，本章一开始介绍 Spring AOP 时，首先介绍的就是这种方式，好处是您可以掌握 Spring AOP 的许多细节，在使用 Spring 2.0 新的 AOP 设置方式时，会容易理解其背后的运作原理。

第二种是基于 XML 的设置。您必须使用基于 Schema 的 XML 设置，并使用 Spring 2.0 新的`<aop>`标签，在设置上会比第一种方式简洁许多，而且 Advice 不用实现特定的接口。

第三种是使用@AspectJ 的 Annotation 支持。由于使用到 Annotation，所以这种方式必须在 JDK 5.0 以上的版本才可以运作，好处是不用在 XML 文件中作繁琐的设置，只要在 Advice 上标记适当的 Annotation 即可完成设置。

4.2 Advices

Advices 实现了 Aspect 的真正逻辑，具体来说在 Java 中就是一个类或更细粒度的设计成一个方法（由一个类来集中管理许多 Advices）。由于织入至 Targets 的时机不同，Spring 提供了几种不同的 Advices，像 Before Advice、After Advice、Around Advice、Throw Advice。在这节中，您可以从编写、使用 Advices 来开始认识 Spring AOP。

4.2.1 Before Advice

Before Advice 会在目标对象的方法执行之前被调用，您可以实现 `org.springframework.aop.MethodBeforeAdvice` 接口来实现 Before Advice 的逻辑，该接口的定义如下所示：

```
package org.springframework.aop;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public interface MethodBeforeAdvice extends BeforeAdvice {
    void before(Method method, Object[] args,
                Object target) throws Throwable;
}
```

在定义中可以看到，MethodBeforeAdvice 继承自 BeforeAdvice 接口，而 BeforeAdvice 接口又继承自 Advice 接口，后两者都是标签接口（Tag interface），只能用作标示而无法定义任何方法，MethodBeforeAdvice 继承了 BeforeAdvice，before() 方法会在目标对象（Target）所指定的方法执行之前被执行，您可以取得被执行的 Method 实例、参数对象数组及目标对象，before() 方法上声明为 void，所以不传回任何的结果，在 before() 方法执行完毕之后，除非您丢出异常，否则目标对象上的方法就会被执行。

以实例来示范如何使用 Before Advice，首先定义目标对象必须实现的接口：

BeforeAdviceDemo — IHello.java

```
package onlyfun.caterpillar;

public interface IHello {
    public void hello(String name);
}
```

接着定义一个 HelloSpeaker 类，让其实现 IHello 接口：

BeforeAdviceDemo — HelloSpeaker.java

```
package onlyfun.caterpillar;

public class HelloSpeaker implements IHello {
    public void hello(String name) {
        System.out.println("Hello, " + name);
    }
}
```

现在 HelloSpeaker 已经编写完毕，在不对它进行任何修改的情况下，您想要在 hello() 方法执行之前，可以记录一些消息，想象一下，这是您拿到的一个组件，您没有源代码，但想对它增加一些日志的服务。您可以实现 MethodBeforeAdvice 接口，例如：

BeforeAdviceDemo — LogBeforeAdvice.java

```
package onlyfun.caterpillar;

import java.lang.reflect.Method;
import java.util.logging.Level;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.util.logging.Logger;
import org.springframework.aop.MethodBeforeAdvice;

public class LogBeforeAdvice
       implements MethodBeforeAdvice {
   private Logger logger =
          Logger.getLogger(this.getClass().getName());

   public void before(Method method, Object[] args,
                 Object target) throws Throwable {
      logger.log(Level.INFO,
            "method starts..." + method);
   }
}
```

在 before()方法的实现中，您加入了一些记录信息的程序代码，LogBeforeAdvice 类被设计为一个独立的服务，可以提供服务给需要的对象，接着只要在定义文件中如下定义：

BeforeAdviceDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

   <bean id="logBeforeAdvice"
        class="onlyfun.caterpillar.LogBeforeAdvice"/>

   <bean id="helloSpeaker"
        class="onlyfun.caterpillar.HelloSpeaker"/>

   <bean id="helloProxy"
      class="org.springframework.aop.framework.ProxyFactoryBean">
       <property name="proxyInterfaces"
               value="onlyfun.caterpillar.IHello"/>
       <property name="target" ref="helloSpeaker"/>
       <property name="interceptorNames">
          <list>
             <value>logBeforeAdvice</value>
          </list>
       </property>
   </bean>

</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意到除了建立 Advice 及 Target 的对象实例之外，您还使用了 org.springframework.aop.framework.ProxyFactoryBean，这个类会被 BeanFactory 或是 ApplicationContext 用来建立代理对象（回忆一下前一节，Spring AOP 主要是通过代理机制来实现的，因而需要建立代理对象），您要在 "proxyInterfaces" 属性上设置代理时可运用的接口，在 "target" 上设置 Target 对象，在 "interceptorNames" 上设置 Advice 实例，在不指定目标方法时，Before Advice 会被织入（Weave）至接口上所有定义的方法之前。

可以编写以下的程序测试一下 Before Advice 的运作：

BeforeAdviceDemo　　SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
    public static void main(String[] args) {
        ApplicationContext context =
               new ClassPathXmlApplicationContext(
                      "beans-config.xml");
        IHello helloProxy =
            (IHello) context.getBean("helloProxy");
        helloProxy.hello("Justin");
    }
}
```

记得在操作取回的代理对象时，必须转换操作接口为 IHello 接口，来看一下这个程序的执行结果，如图 4.7 所示。

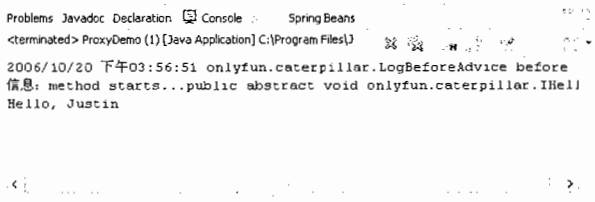


图 4.7　BeforeAdviceDemo 项目的执行结果

在这个范例中可以看到，您所设计的 HelloSpeaker 与 LogBeforeAdvice 是两个独立的对象，对于 HelloSpeaker 来说，它不用知道 LogBeforeAdvice 的存在（也就是没有任

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

何与 LogBeforeAdvice 相关的 API 编写在 HelloSpeaker 中），而 LogBeforeAdvice 也可以运用至其他的对象之上，HelloSpeaker 与 LogBeforeAdvice 都是可以重复使用的设计。

从以上范例可以看出 AOP 的精神，着重于 Aspects 的辨识，设计可重复使用的 Advices，就如 OOP 重视对象的辨识，设计可重复使用的对象一样。

良葛格的话匣子<<<

如果您逐一将 .jar 文件加入至项目管理之中，记得将 spring-core.jar、spring-beans.jar、spring-context.jar、spring-aop.jar 与 commons-logging.jar 等文件加入 Classpath。

4.2.2 After Advice

After Advice 会在目标方法执行之后被调用，您可以实现 org.springframework.aop.AfterReturningAdvice 接口来实现 After Advice 的逻辑，AfterReturningAdvice 接口的定义如下：

```
package org.springframework.aop;

public interface AfterReturningAdvice extends Advice {
    void afterReturning(Object returnValue, Method m,
            Object[] args, Object target) throws Throwable;
}
```

AfterReturningAdvice 直接继承自 Advice 接口，afterReturning() 当中传入的参数有目标方法的回传值、方法实例、参数对象数组与目标对象，afterReturning() 方法传回 void，若要中止接下来的应用程序流程，丢出异常是唯一的方式。

可以在 Before Advice 介绍的例子当中，在 HelloSpeaker 的 hello() 方法执行之后，加上 AfterReturningAdvice，首先定义一个 LogAfterAdvice 类来实现 AfterReturningAdvice 接口：

AfterAdviceDemo　　LogAfterAdvice.java

```
package onlyfun.caterpillar;

import java.lang.reflect.Method;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.springframework.aop.AfterReturningAdvice;

public class LogAfterAdvice
                implements AfterReturningAdvice {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    public void afterReturning(Object object,
                               Method method,
                               Object[] args,
                               Object target) throws Throwable {
        logger.log(Level.INFO, "method ends..." + method);
    }
}
```

接着只要在 *beans-config.xml* 中增加 After Advice 的实例，以及在 ProxyFactoryBean 的 "interceptorNames"中增加对 LogAfterAdvice 的引用，定义文件编写如下：

AfterAdviceDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="logBeforeAdvice"
         class="onlyfun.caterpillar.LogBeforeAdvice"/>

    <bean id="logAfterAdvice"
         class="onlyfun.caterpillar.LogAfterAdvice"/>

    <bean id="helloSpeaker"
         class="onlyfun.caterpillar.HelloSpeaker"/>

    <bean id="helloProxy"
       class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces"
                 value="onlyfun.caterpillar.IHello"/>
        <property name="target" ref="helloSpeaker"/>
        <property name="interceptorNames">
            <list>
                <value>logBeforeAdvice</value>
                <value>logAfterAdvice</value>
            </list>
        </property>
    </bean>

</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在定义中，除了之前设置的 LogBeforeAdvice 之外，还加入了 LogAfterAdvice 的日志服务，其他的程序与 BeforeAdviceDemo 项目相同，执行时的一个参考画面如图 4.8 所示：

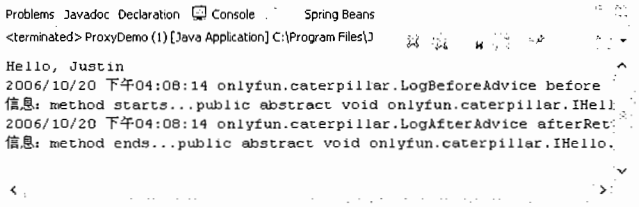


图 4.8 AfterAdviceDemo 项目的执行结果

良葛格的话匣子<<<

您也可以设计一个 LogAspect 类，让它同时实现 MethodBeforeAdvice 与 AfterReturningAdvice 接口，本节的范例为了强调接口的实现，因而将 Advice 分开设计。

4.2.3 Around Advice

在 Before Advice 与 After Advice 的介绍中，您已经知道如何在目标对象的方法执行前、执行后介入 Advices 的服务逻辑。事实上如果您要在方法执行前后加入 Advices 的服务逻辑，您可以直接通过实现 org.aopalliance.intercept.MethodInterceptor 接口，于方法执行前、执行后执行相关的服务，而不用分别提供 Before Advice 与 After Advice，MethodInterceptor 接口的定义如下：

```
package org.aopalliance.intercept;

public interface MethodInterceptor {
    public Object invoke(
        MethodInvocation methodInvocation) throws Throwable;
}
```

注意到 MethodInterceptor 的包名称是 org.aopalliance.intercept，可以得知这个接口是由 AOP Alliance 所制订的，这表示实现 MethodInterceptor 接口的类，将可以相容于遵守 AOP Alliance 规范的 AOP 框架。

与 Before Advice 及 After Advice 不同的是，在 MethodInterceptor 的 invoke() 方法中，您要自行决定是否使用 org.aopalliance.intercept. MethodInvocation 的 proceed() 方法来执行目标对象的方法。proceed() 会回传方法执行后的 Object 执行结果，所以在 invoke() 结束之前，您有机会修改这个对象，或是回传另一个完全不相干的对象，当然了，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

只有在真正必要的时候才会这么做。

实际来看看如何实现 MethodInterceptor，可以直接在之前的 AfterAdviceDemo 项目进行编写：

AroundAdviceDemo LogInterceptor.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class LogInterceptor implements MethodInterceptor {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

   public Object invoke(MethodInvocation methodInvocation)
                                        throws Throwable {
        logger.log(Level.INFO,
           "method starts..." + methodInvocation.getMethod());

        Object result = null;

        try {
          result = methodInvocation.proceed();
        }
        finally {
            logger.log(Level.INFO,
               "method ends..." +
               methodInvocation.getMethod() + "\n");
        }

        return result;
   }
}
```

可以用这个 Interceptor 来取代 AfterAdviceDemo 项目中的 LogBeforeAdvice 与 LogAfterAdvice，Bean 定义文件的编写方式如下所示：

AroundAdviceDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

      <bean id="logInterceptor"
           class="onlyfun.caterpillar.LogInterceptor"/>

      <bean id="helloSpeaker"
           class="onlyfun.caterpillar.HelloSpeaker"/>

      <bean id="helloProxy"
          class="org.springframework.aop.framework.ProxyFactoryBean">
          <property name="proxyInterfaces"
                   value="onlyfun.caterpillar.IHello"/>
          <property name="target" ref="helloSpeaker"/>
          <property name="interceptorNames">
              <list>
                  <value>logInterceptor</value>
              </list>
          </property>
      </bean>

</beans>
```

执行结果与 AfterAdviceDemo 项目的执行结果是一样的，可以参考图 4.5 的画面。

Spring 在真正执行某个方法前，会先插入 Interceptor，如果有多个 Interceptor，每个 Interceptor 会执行自己的处理，然后执行 MethodInvocation 的 proceed()方法，这将执行流程转给下一个 Interceptor，如果没有下一个 Interceptor，就执行真正调用的方法，方法执行完毕后，再一层一层返回 Interceptor 堆栈，最后离开堆栈返回应用程序本身的流程。

4.2.4 Throw Advice

若想要在异常发生时通知某些服务对象作某些事，您可以使用 Throws Advice，在 Spring 中想要实现 Throws Advice，必须实现 org.springframework.aop.ThrowsAdvice 接口，然而这个接口并没有定义任何的方法，它只是一个标签接口（Tag interface），您可以在当中定义 afterThrowing 方法名称，只要它是以下的形式：

```
afterThrowing([Method], [args], [target], subclassOfThrowable);
```

方括号 [] 中的设置，例如 Method、args 与 target 等表示是可以省略的，方法中一定要

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的是 subclassOfThrowable，也就是这个参数必须是 Throwable 的子类，在异常发生时，会检验所设置的 Throws Advice 中是否有符合异常类型的方法，如果有的话就通知它执行，以下是两个方法声明的例子：

```
void afterThrowing(Throwable throwable);
void afterThrowing(Method method, Object[] args,
                   Object target, Throwable throwable);
```

在方法上如果声明了不同的 Throwable 类型，则按照异常发生类型的不同，通知不同的方法，例如 SomeException 会通知声明有 SomeException 参数的方法，而 OtherException 会通知声明有 OtherException 的方法。

注意到当异常发生时，Throw Advice 的任务只是执行对应的方法，您并不能在 Throws Advice 中将异常处理掉，在 Throw Advice 执行完毕后，原先的异常仍被传播至应用程序之中，Throw Advice 并不介入应用程序的异常处理，异常处理仍旧是应用程序本身所要负责的，如果想要在 Throw Advice 处理时中止应用程序的处理流程，作法是丢出其他的异常。

来看个 Throws Advice 的实际例子，首先定义 IHello 接口：

ThrowAdviceDemo — IHello.java

```
package onlyfun.caterpillar;

public interface IHello {
    public void hello(String name) throws Throwable;
}
```

接着定义一个 HelloSpeaker 类来实现 IHello 接口，并在 hello() 方法当中模拟程序发生错误时的异常丢出：

ThrowAdviceDemo — HelloSpeaker.java

```
package onlyfun.caterpillar;

public class HelloSpeaker implements IHello {
    public void hello(String name) throws Throwable {
        System.out.println("Hello, " + name);
        // 抱歉！程序错误！发生异常 XD
        throw new Exception("发生异常...");
    }
}
```

如果您需要在应用程序丢出异常时，介入 Throw Advice 提供一些服务，例如记录一些异常信息，则可以实现 ThrowAdvice 接口：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ThrowAdviceDemo SomeThrowAdvice.java

```
package onlyfun.caterpillar;

import java.lang.reflect.Method;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.springframework.aop.ThrowsAdvice;

public class SomeThrowAdvice implements ThrowsAdvice {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());
    public void afterThrowing(Method method, Object[] args,
                              Object target, Throwable subclass) {
        // 记录异常
        logger.log(Level.INFO,
                "Logging that a " + subclass +
                "Exception was thrown in " + method);

    }
}
```

接着在 Bean 定义文件中写下以下的定义，让 Throw Advice 在异常发生时提供日志服务：

ThrowAdviceDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someThrowAdvice"
          class="onlyfun.caterpillar.SomeThrowAdvice"/>

    <bean id="helloSpeaker"
          class="onlyfun.caterpillar.HelloSpeaker"/>

    <bean id="helloProxy"
         class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces"
                  value="onlyfun.caterpillar.IHello"/>
        <property name="target" ref="helloSpeaker"/>
        <property name="interceptorNames">
            <list>
                <value>someThrowAdvice</value>
            </list>
        </property>
    </bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
</beans>
```

可以编写以下的程序来测试一下 Throw Advice 是否如预期般运作：

ThrowAdviceDemo　　SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
   public static void main(String[] args) {
      ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "beans-config.xml");
      IHello helloProxy =
         (IHello) context.getBean("helloProxy");

      try {
         helloProxy.hello("Justin");
      }
      catch(Throwable throwable) {
         // 应用程序的异常处理
         System.err.println(throwable);
      }
   }
}
```

在程序中，应用程序本身定义了异常处理的逻辑，Throw Advice 不介入应用程序处理异常的逻辑，Throw Advice 提供额外的日志服务，来看看一个执行时的参考画面，如图 4.9 所示。

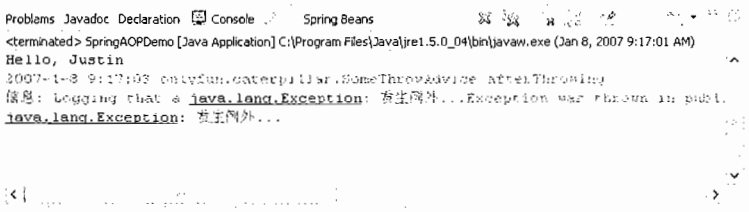


图 4.9　ThrowAdviceDemo 项目的执行结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.3 Pointcut、Advisor

在前一节中，所定义的 Advice 都是直接织入至代理的接口执行前后的，或者在执行方法过程中异常发生时织入。事实上还可以定义更细部的织入时机。Pointcut 定义了感兴趣的 Jointpoint（Advice 的应用时机）。在 Spring 中，使用 PointcutAdvisor 提供 Pointcut 实例，具体结合 Advice，Spring 内建的 Pointcut 都有对应的 PointcutAdvisor。

4.3.1 NameMatchMethodPointcutAdvisor

Pointcut 定义了 Jointpoint，在 Spring 中使用 PointcutAdvisor 来提供 Pointcut、结合 Advice，PointcutAdvisor 为 Advisor 的子接口，Advisor 接口的定义如下：

```
package org.springframework.aop;

import org.aopalliance.aop.Advice;

public interface Advisor {
    boolean isPerInstance();
    Advice getAdvice();
}
```

PointcutAdvisor 接口在 Spring 中的定义如下：

```
package org.springframework.aop;

public interface PointcutAdvisor extends Advisor {
    Pointcut getPointcut();
}
```

Spring 中内建的 Pointcut 都有对应的 PointcutAdvisor，在这里先来介绍一下，如何使用 Spring 所提供的 org.springframework.aop.support.NameMatchMethodPointcutAdvisor，这是最基本的 PointcutAdvisor，用以提供 Spring 中静态的 Pointcut 实例，可使用表示式（expression）指定 Advice 应用目标上的方法名称，或者是用 * 来指定，例如当 hello* 表示执行代理对象上以 hello 作为开头的方法名称时，它都会应用指定的 Advices（在这个主题之前的例子，Advice 会被应用至所有代理的方法）。

举个实际的例子来说，假设您定义了 IHello 的接口：

NameMatchDemo　　IHello.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public interface IHello {
    public void helloNewbie(String name);
    public void helloMaster(String name);
}
```

接着定义 HelloSpeaker 类来实现 IHello 接口：

NameMatchDemo HelloSpeaker.java

```
package onlyfun.caterpillar;

public class HelloSpeaker implements IHello {
    public void helloNewbie(String name) {
        System.out.println("Hello, " + name + " newbie!");
    }

    public void helloMaster(String name) {
        System.out.println("Hello, " + name  + " master!");
    }
}
```

接着可以编写一个简单的 Advice，例如这里会使用到 BeforeAdviceDemo 项目中的 LogBeforeAdvice，基于篇幅有限，请参考 4.2.1 节中的 LogBeforeAdvice 类的编写，这里不再重复列出程序代码。

接着编写以下的 Bean 定义文件，使用 NameMatchMethodPointcutAdvisor 提供 Pointcut 实例，并将 Advice 结合在一起：

NameMatchDemo beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="logBeforeAdvice"
          class="onlyfun.caterpillar.LogBeforeAdvice"/>

    <bean id="helloAdvisor"
          class="org.springframework.aop.
                 → support.NameMatchMethodPointcutAdvisor">
       <property name="mappedName" value="*Newbie"/>
       <property name="advice" ref="logBeforeAdvice"/>
    </bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="helloSpeaker"
         class="onlyfun.caterpillar.HelloSpeaker"/>

    <bean id="helloProxy"
       class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces"
                 value="onlyfun.caterpillar.IHello"/>
        <property name="target" ref="helloSpeaker"/>
        <property name="interceptorNames">
            <list>
                <value>helloAdvisor</value>
            </list>
        </property>
    </bean>

</beans>
```

在 NameMatchMethodPointcutAdvisor 的 "mappedName" 属性上，由于指定了"*Newbie"，所以执行 helloNewbie()方法时，由于方法名称符合 "*Newbie"，就会应用"logBeforeAdvice"的服务逻辑，可以编写以下的程序来进行测试，看看结果是否符合预期：

NameMatchDemo　　SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
    public static void main(String[] args) {
        ApplicationContext context =
               new ClassPathXmlApplicationContext(
                      "beans-config.xml");

        IHello helloProxy =
            (IHello) context.getBean("helloProxy");

        helloProxy.helloNewbie("Justin");
        helloProxy.helloMaster("caterpillar");
    }
}
```

执行时的一个结果参考画面如图 4.10 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

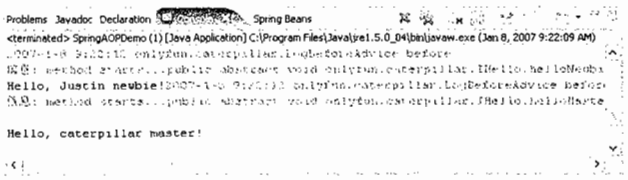


图 4.10　NameMatchDemo 项目的执行结果

您也可以在定义 Advisor 时，使用 "mappedNames" 属性进行多个 Pointcut 表示式定义，例如可修改 *beans-config.xml* 为以下的内容：

```
<beans ...>
    ....
    <bean id="helloAdvisor"
         class="org.springframework.aop.
                → support.NameMatchMethodPointcutAdvisor">
       <property name="mappedNames">
          <list>
             <value>helloNewbie</value>
             <value>helloMaster</value>
          </list>
       </property>
       <property name="advice" ref="logBeforeAdvice"/>
    </bean>

    ....
</beans>
```

4.3.2　RegExpMethodPointcutAdvisor

Spring 提供的 `org.springframework.aop.support.RegexpMethodPointcutAdvisor` 可以让您使用 Regular expression 来编写 Pointcut 表示式，用以提供 Spring 中静态 Pointcut 的实例。在符合 Regular expression 的情况下应用 Advices，您可以使用以下的几个符号：

表 4.1　RegExpMethodPointcutAdvisor 定义时可用的符号

符号	描述
.	符合任何单一字符
+	符合前一个字符一次或多次
*	符合前一个字符零次或多次
\	Escape 任何 Regular expression 使用到的符号

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

RegexpMethodPointcutAdvisor 的 "pattern" 属性让您指定要符合的完整类名称（包括包名称）和方法名称，例如若要求符合 onlyfun.caterpillar.IHello 下的 hello 开始的方法名称，则要如下编写：

```
onlyfun\.caterpillar\.IHello\.hello.*
```

由于“.”符号已经被 Regular expression 使用，所以表示式中要指定“.”符号，则要转义（Escape），也就是使用“\.”的方式，如果只打算针对方法名称比对，而不管包名称，则可以这么编写：

```
.*hello.*
```

可以使用 NameMatchDemo 项目中所使用的类与 Advice，只要改一下 Bean 定义文件就可以了，例如：

RegexpMatchDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="logBeforeAdvice"
         class="onlyfun.caterpillar.LogBeforeAdvice"/>

    <bean id="regExpAdvisor"
         class="org.springframework.aop.
             → support.RegexpMethodPointcutAdvisor">
       <property name="pattern" value=".*Newbie"/>
       <property name="advice" ref="logBeforeAdvice"/>
    </bean>

    <bean id="helloSpeaker"
         class="onlyfun.caterpillar.HelloSpeaker"/>

    <bean id="helloProxy"
       class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces"
                 value="onlyfun.caterpillar.IHello"/>
        <property name="target" ref="helloSpeaker"/>
        <property name="interceptorNames">
            <list>
                <value>regExpAdvisor</value>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            </list>
        </property>
    </bean>
</beans>
```

在上面的定义中，无论其前面的包、类名称是怎样的，设置符合方法名称为 Newbie 结尾的方法都要应用 Advice，所以执行结果与 NameMatchDemo 项目的执行结果是一样的。

4.3.3 ControlFlowPointcut

org.springframework.aop.support.ControlFlowPointcut 是 Sping 所提供的类，作用是判断在方法的执行堆栈中，某个指定类的某方法是否曾经要求您的目标对象执行某个动作，由于这在执行时期才会确定是否介入 Advices，所以 Spring 提供的是动态 Pointcut 功能。

以 NameMatchMethodPointcutAdvisor 中的 LogBeforeAdvice 类为例，您想要知道在 onlyfun.caterpillar.Some 类中，是否在某个方法中要求过指定的目标对象执行某些动作，如果有的话，则介入 LogBeforeAdvice 来提供日志服务，可以将 Bean 定义文件如下编写：

ControlFlowDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="some" class="onlyfun.caterpillar.Some"/>

    <bean id="logBeforeAdvice"
         class="onlyfun.caterpillar.LogBeforeAdvice"/>

    <bean id="helloFlowControlPointcut"
       class="org.springframework.aop.support.ControlFlowPointcut">
        <constructor-arg value="onlyfun.caterpillar.Some"/>
    </bean>

    <bean id="helloAdvisor"
     class="org.springframework.aop.support.DefaultPointcutAdvisor">
        <property name="advice" ref="logBeforeAdvice"/>
        <property name="pointcut" ref="helloFlowControlPointcut"/>
    </bean>

    <bean id="helloSpeaker"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        class="onlyfun.caterpillar.HelloSpeaker"/>

    <bean id="helloProxy"
      class="org.springframework.aop.framework.ProxyFactoryBean">
       <property name="proxyInterfaces"
               value="onlyfun.caterpillar.IHello"/>
       <property name="target" ref="helloSpeaker"/>
       <property name="interceptorNames">
          <list>
             <value>helloAdvisor</value>
          </list>
       </property>
    </bean>

</beans>
```

在 ControlFlowPointcut 构造时，指定了 onlyfun.caterpillar.Some 类，表示若在 Some 类中的某个方法要求指定的目标对象（也就是"helloSpeaker"实例）执行了某些动作，则应用 Before Advice（logBeforeAdvice）提供日志的服务，Some 类如下编写：

ControlFlowDemo　　Some.java

```
package onlyfun.caterpillar;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;

public class Some implements ApplicationContextAware {
    private IHello helloProxy;

    public void setApplicationContext(
           ApplicationContext context) throws BeansException {
        helloProxy = (IHello) context.getBean("helloProxy");
    }

    public void helloEverybody() {
        helloProxy.helloNewbie("Justin");
        helloProxy.helloMaster("caterpillar");
    }

    public void hello() {
        System.out.println("Hello!");
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

为了方便取得 ApplicationContext 以获得 helloSpeaker 的代理对象，Some 类实现了 org.springframework.context.ApplicationContextAware 接口，接着可以编写一个简单的程序来测试一下 ControlFlowPointcut 的运作，如下所示：

ControlFlowDemo　　SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
    public static void main(String[] args) {
        ApplicationContext context =
               new ClassPathXmlApplicationContext(
                      "beans-config.xml");

        Some some = (Some) context.getBean("some");

        if(args.length > 0 && "run".equals(args[0])) {
            some.helloEverybody();
        }
        else {
            some.hello();
        }
    }
}
```

其余未列出的程序，都与 NameMatchDemo 项目中的程序相同，这里就不再列出，完整的程序内容可以参考 ControlFlowDemo 项目，来看一下执行的结果。

如果执行程序时没有提供 "run" 参数，则只会出现 "Hello!" 的文字消息，这是因为没有执行 Some 实例的 helloEverybody()，在执行堆栈中并不符合所指定的定义：Some 类的某方法曾要求"helloSpeaker"实例执行某些动作。

如果执行程序时提供了 "run" 参数，则会执行 Some 类的 helloEverybody() 方法，方法中要求"helloSpeaker"的代理对象执行 helloNewbie() 与 helloMaster() 方法，符合 Bean 定义文件中指定的内容，因而会应用 LogBeforeAdvice 来提供服务消息，一个执行的结果如图 4.11 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

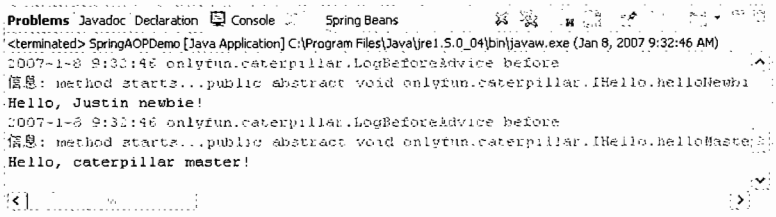


图 4.11 ControlFlowDemo 项目的执行结果

动态 Pointcut 的问题就是在于性能上的付出，由于执行堆栈的判断是在执行时期进行的，所以执行时会很慢，在不同的 JDK 上可能会有 5 到 10 倍的性能延迟，因此建议在可能的情况下，尽量使用静态 Pointcut。

良葛格的话匣子<<<

在 Eclipse 中要指定执行时的参数，可以执行 Eclipse 菜单上的“Run/Run...”指令，然后选择“Arguments”页签，在“Program arguments”文字框中输入参数，如图 4.12 所示。

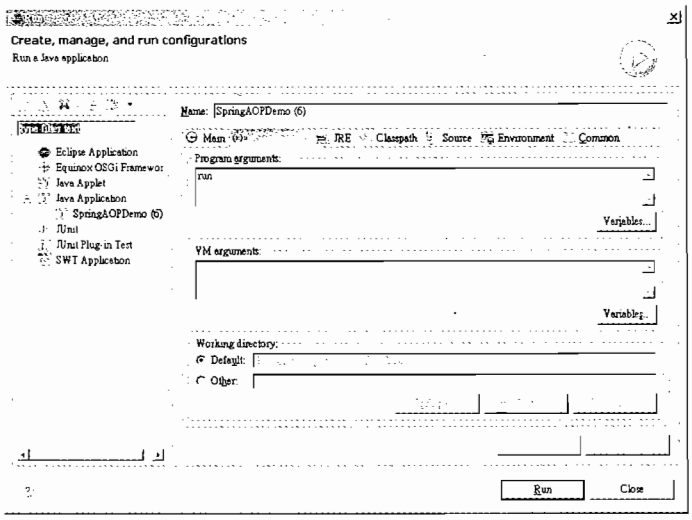


图 4.12 在 Eclipse 中指定执行时的参数

4.3.4 Pointcut 接口

Spring 的 Pointcut 是通过实现 `org.springframework.aop.Pointcut` 接口来实现的，其定义如下：

```
package org.springframework.aop;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public interface Pointcut {
    ClassFilter getClassFilter();
    MethodMatcher getMethodMatcher();
    Pointcut TRUE = TruePointcut.INSTANCE;
}
```

Pointcut.TRUE 是 Pointcut 接口的简单实现，它传回的 ClassFilter 是 ClassFilter.TRUE，而传回的 MethodMatcher 是传回 MethodMatcher.TRUE。

ClassFilter 接口决定一个类是否要应用 Advice，其定义如下所示：

```
package org.springframework.aop;

public interface ClassFilter {
    boolean matches(Class clazz);
    ClassFilter TRUE = TrueClassFilter.INSTANCE;
}
```

matches() 方法要决定传入的类是不是符合 Pointcut 的定义，ClassFilter.TRUE 是 ClassFilter 接口的简单实现，它的 matches() 方法总是传回 true，如果想要建立的 Pointcut 只考虑到方法名称，则可以使用这个方法。

而 MethodMatcher 决定某个方法是否要应用 Advice，其定义如下所示：

```
package org.springframework.aop;

import java.lang.reflect.Method;

public interface MethodMatcher {
    boolean matches(Method method, Class targetClass);
    boolean isRuntime();
    boolean matches(Method method,
                    Class targetClass, Object[] args);
    MethodMatcher TRUE = TrueMethodMatcher.INSTANCE;
}
```

matches() 方法决定某个类的某个方法是否符合 Pointcut 定义，有两个版本，第一个版本使用于静态 Pointcut，像 NameMatchMethodPointcutAdvisor、RegExpMethodPointcutAdvisor，第一个方法总是会被执行，如果是静态 Pointcut，则 isRuntime() 会传回 false，此时第二个 matches() 方法不会被执行，只有在 isRuntime() 为 true 时，第二个版本的 matches() 才会被执行，例如 ControlFlowPointcut。

MethodMatcher.TRUE 是 MethodMatcher 的简单实现，它第一个版本的 matches() 总是

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

传回 true，isRuntime()总是传回 false，表示静态 Pointcut，所以不可以执行第二个版本的 matches() 方法，否则就会丢出 UnsupportedOperationException 异常。

4.3.5 Pointcut 交集、并集操作

如果建立了多个 Pointcuts，并且在这之中，您发现有几个 Pointcuts 的定义可以组合在一起，成为另一个 Pointcuts 的定义时，您可以直接对现在的 Pointcuts 对象，进行交集（Intersection）、并集（Union）的操作，org.springframework.aop.support.ComposablePointcut 是 Pointcut 的实现类，可以操作它的 intersection()、union()方法来完成这个操作：

```
ComposablePointcut intersection(ClassFilter filter);
ComposablePointcut intersection(MethodMatcher mm)
ComposablePointcut intersection(Pointcut other)
ComposablePointcut union(ClassFilter filter)
ComposablePointcut union(MethodMatcher mm)
```

例如：

```
Pointcut pc = new ComposablePointcut()
                .union(classFilter)
                .intersection(methodMatcher)
                .intersection(pointcut);
```

在 ComposablePointcut 中没有方法可以处理两个 Pointcut 的并集，您得使用 org.springframework.aop.support.Pointcuts 的 union() 方法，例如：

```
Pointcut union = Pointcuts.union(pointcut1, pointcut2);
```

当然了，这必须是在程序中直接编写才可以进行交集或并集操作，如果您希望在 Bean 定义文件中直接声明，则可以设计一个类，然后于 Bean 定义文件中声明该类的实例，这可以增加 Pointcut 操作的重用性，例如：

```
package onlyfun.caterpillar;

import java.util.List;

import org.springframework.aop.ClassFilter;
import org.springframework.aop.MethodMatcher;
import org.springframework.aop.Pointcut;
import org.springframework.aop.
                    framework.AopConfigException;
import org.springframework.aop.support.Pointcuts;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class UnionPoint implements Pointcut {
    private Pointcut pointcut;

    public ClassFilter getClassFilter() {
        return getPointcut().getClassFilter();
    }

    public MethodMatcher getMethodMatcher() {
        return getPointcut().getMethodMatcher();
    }

    public void setPointcuts(List pointcuts) {
        if(pointcuts == null || pointcuts.size == 0) {
            throw new AopConfigException(
                        "至少要设置一个 Pointcut");
        }

        pointcut = (Pointcut) pointcuts.get(0);

        for(int i = 0; i < pointcuts.size(); i++) {
             Pointcut next = (Pointcut) pointcuts.get(i);
             pointcut = Pointcuts.union(pointcut, next);
        }
    }

    private Pointcut getPointcut() {
        if(pointcut == null) {
            throw new AopConfigExeption("没有设置 Pointcut");
        }
         return pointcut;
    }
}
```

4.4 Introduction

在 Spring 中，Introduction 是一种特殊的 Advice，从行为上来看，它不像 Before Advice、After Advice 等 Advice 在方法前后介入服务，而是直接介入整个对象的行为，就好像对象凭空多了一些可操作的行为，为对象动态加入（Mixin）原先所没有的职责。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.4.1 IntroductionInterceptor

对于之前介绍过的 Before Advice、After Advice、Around Advice、Throw Advice，从用户的角度来看，它们“影响了目标对象上某些方法的行为”，例如让某些方法看来似乎增加了一些日志的动作。

Introduction 是个特别的 Advice，从用户的角度来看，它“影响了目标对象的行为定义，直接增加了目标对象的职责（具体来说就是增加了可操作的方法）”，例如让某个已定义好的对象，在不修改该对象的类文件的情况下，却可以增加一些额外的操作方法到对象之上。

就 Java 程序语言类设计的观点来说，动态地为对象增加可操作的方法显得不可思议，事实上在 Spring AOP 中，可以通过实现 `org.springframework.aop.IntroductionInterceptor` 来实现 Introduction。

良葛格的话匣子<<<

熟悉动态脚本语言的设计人员，例如熟悉 JavaScript 的设计人员，对动态为对象增加可操作的方法（函数）来说，则是很习以为常的功能。

`IntroductionInterceptor` 继承了 `MethodInterceptor` 与 `DynamicIntroductionAdvice` 接口，其中 `implementsInterface()` 方法（继承自 DynamicIntroductionAdvice）如果传回 `true`，表示目前的 `IntroductionInterceptor` 实现了规定的接口（也就是要额外增加行为的接口），此时您要使用 `invoke()` 执行接口上的方法，让目标对象执行额外的行为，您不可使用 `MethodInvocation` 的 `proceed()` 方法，因为要执行的是对象上原来没有的行为，执行 `proceed()` 方法没有意义。

从文字上来理解 `Introduction` 会比较抽象，举个实际的例子来说，假设您的系统中已经有以下的类：

IntroductionDemo — ISome.java

```
package onlyfun.caterpillar;

public interface ISome {
    public void doSome();
}
```

IntroductionDemo — Some.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class Some implements ISome {
    public void doSome() {
        System.out.println("原来对象的职责。。。");
    }
}
```

您希望在不修改原始文件的情况下，为 Some 类增加一些可操作的方法，也许您甚至连源代码文件都没有，只有*.class* 文件，唯一知道的也许是它们的 API 说明，在不对它们作出修改的情况下，希望 Some 类可以增加 doOther()方法。

在 Spring 中，可以通过实现 IntroductionInterceptor 接口来完成上面的任务，首先为 doOther()方法建立接口：

IntroductionDemo　　IOther.java

```
package onlyfun.caterpillar;

public interface IOther {
    public void doOther();
}
```

接着定义一个 OtherIntroduction 类同时实现 IntroductionInterceptor 接口与 IOther 接口，例如：

IntroductionDemo　　OtherIntroduction.java

```
package onlyfun.caterpillar;

import org.aopalliance.intercept.MethodInvocation;
import org.springframework.aop.IntroductionInterceptor;

public class OtherIntroduction
            implements IntroductionInterceptor, IOther {
    // 是否实现自 IOther 接口
    public boolean implementsInterface(Class clazz) {
        return clazz.isAssignableFrom(IOther.class);
    }

    public Object invoke(MethodInvocation methodInvocation)
                                throws Throwable {
        // 如果执行的方法来自 IOther 接口的定义
        if(implementsInterface(
            methodInvocation.getMethod().getDeclaringClass())) {
            // 执行额外加入（mixin）的行为
            return methodInvocation.getMethod().
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                invoke(this, methodInvocation.getArguments());
        }
        else {
            return methodInvocation.proceed();
        }
    }

    public void doOther() {
        System.out.println("增加的职责...");
    }
}
```

接着要在 Bean 定义文件中将 Introduction 织入至 Some 对象之上，使用 org.springframework.aop.support.DefaultIntroductionAdvisor 就可以了，例如：

IntroductionDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="some"
         class="onlyfun.caterpillar.Some"/>

    <bean id="otherIntroduction"
         class="onlyfun.caterpillar.OtherIntroduction"/>

    <bean id="otherAdvisor"
         class="org.springframework.aop.
                → support.DefaultIntroductionAdvisor">
         <constructor-arg ref="otherIntroduction"/>
         <constructor-arg value="onlyfun.caterpillar.IOther"/>
    </bean>

    <bean id="proxyFactoryBean"
       class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces"
                value="onlyfun.caterpillar.ISome"/>
        <property name="target" ref="some"/>
        <property name="interceptorNames">
            <list>
                <value>otherAdvisor</value>
            </list>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        </property>
    </bean>

</beans>
```

DefaultIntroductionAdvisor 在构造时，需要给它 IntroductionInterceptor 的实例，以及所要代理额外行为的接口，现在，来编写一个简单的程序测试，从这个程序当中，可以更进一步了解什么是为对象额外增加行为：

IntroductionDemo　　SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
            support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                    "beans-config.xml");

        ISome some =
            (ISome) context.getBean("proxyFactoryBean");

        some.doSome();

        // 看来好像 some 对象动态增加了职责
        ((IOther) some).doOther();
    }
}
```

对于 some 所参考的对象来说，它原先并不会有 doOther() 方法可供操作，然而通过 Spring AOP 的 Introduction 机制，现在 some 所参考的对象多了 doOther() 方法可以操作，可以来看一下执行的结果，如图 4.13 所示。

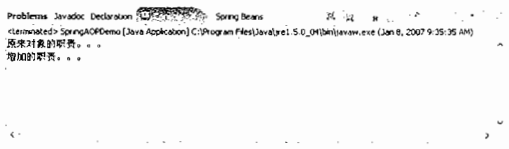


图 4.13　IntroductionDemo 项目的执行结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.4.2 DelegatingIntroductionInterceptor

org.springframework.aop.support.DelegatingIntroductionInterceptor 是 Spring AOP 中为 IntroductionInterceptor 接口所提供的实现类。您可以直接继承这个类，添加希望为目标对象增加的行为，并可以有对象自己的状态，例如让对象携带有“是”、“否”锁定的状态，DelegatingIntroductionInterceptor 已经实现了大部分的细节。

举个例子来说，假设系统中已经有这样的类：

DelegatingIntroductionDemo ISome.java

```
package onlyfun.caterpillar;

public interface ISome {
    public void setSome(String some);
    public String getSome();
}
```

DelegatingIntroductionDemo Some.java

```
package onlyfun.caterpillar;

public class Some implements ISome {
    private String some;

    public void setSome(String some) {
        this.some = some;
    }

    public String getSome() {
        return some;
    }
}
```

在不修改 *Some.java* 程序内容的情况下，您希望增加一个 locked 的 boolean 类型数据成员，并增加可操作的 lock()与 unlock()方法来设置 locked 成员为 true 或 false，如果 locked 被设置为 true，则锁定 setSome()方法无法被执行，也就是将对象锁定为不可变动（Immutable）。

可以先定义一个 ILockable 接口，上面定义的是想添加至目标对象的操作方法：

DelegatingIntroductionDemo ILockable.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public interface ILockable {
    public void lock();
    public void unlock();
    public boolean isLocked();
}
```

接着继承 DelegatingIntroductionInterceptor 类，并同时实现 ILockable 接口：

DelegatingIntroductionDemo　　LockIntroduction.java

```
package onlyfun.caterpillar;

import org.springframework.aop.
           support.DelegatingIntroductionInterceptor;
import org.aopalliance.intercept.MethodInvocation;
import org.springframework.aop.
           framework.AopConfigException;

public class LockIntroduction
            extends DelegatingIntroductionInterceptor
            implements ILockable {
    private boolean locked;

    public Object invoke(MethodInvocation invocation)
                    throws Throwable {
        // locked 为 true 则不能执行 set 方法
        if (isLocked() &&
            invocation.getMethod().
                    getName().indexOf("set") == 0) {
            throw new AopConfigException(
                            "对象被锁定！！");
        }

        return super.invoke(invocation);
    }

    public void lock() {
        locked = true;
    }

    public void unlock() {
        locked = false;
    }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public boolean isLocked() {
        return locked;
    }
}
```

新增的行为是，当对象使用 lock()方法设置 locked 为 true 时锁定对象，如果此时有其他对象打算执行 set 方法则丢出异常，通知调用者对象已处于锁定状态，至于 Bean 定义文件的内容，可以如下编写：

DelegatingIntroductionDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="some"
         class="onlyfun.caterpillar.Some"/>

    <bean id="lockIntroduction"
         class="onlyfun.caterpillar.LockIntroduction"/>

    <bean id="lockAdvisor"
       class="org.springframework.aop.
                  → support.DefaultIntroductionAdvisor">
         <constructor-arg ref="lockIntroduction"/>
         <constructor-arg value="onlyfun.caterpillar.ILockable"/>
    </bean>

    <bean id="proxyFactoryBean"
       class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces"
                 value="onlyfun.caterpillar.ISome"/>
        <property name="target" ref="some"/>
        <property name="interceptorNames">
            <list>
                <value>lockAdvisor</value>
            </list>
        </property>
    </bean>

</beans>
```

来编写一个简单的测试程序，看看如何利用增加的行为来进行对象锁定：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

DelegatingIntroductionDemo　　SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
    public static void main(String[] args) throws Exception {
        ApplicationContext context =
               new ClassPathXmlApplicationContext(
                      "beans-config.xml");

        ISome some =
            (ISome) context.getBean("proxyFactoryBean");

        // 对象没有被锁定，可以执行 set 方法
        some.setSome("justin");
        System.out.println(some.getSome());

        try {
            // 对象被锁定
            ((ILockable) some).lock();

            // 无法执行 set 方法，丢出异常
            some.setSome("momor");
            // 由于会丢出异常，所以下面的这行程序无法被执行
            System.out.println(some.getSome());
        }
        catch(Throwable e) {
            e.printStackTrace();
        }

        // Object is unlocked.
        ((ILockable) some).unlock();
        // It's ok to use setter again.
        some.setSome("momor");
        System.out.println(some.getSome());
    }
}
```

在 some 所参考的对象上，可以执行新添加的 lock()方法来进行锁定，当 some 所参考的对象被锁定时，执行 set 方法则会丢出异常，可以执行 unlock()方法解除锁定，一个执行结果如图 4.14 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 4.14 DelegatingIntroductionDemo 项目的执行结果

事实上，Some 类并没有真正增加行为，从以上两个项目的例子中可以看出，Introduction 事实上是利用委托的方式，当执行非 Some 类上所定义的方法时，在代理对象中再委托 Introduction 对象来执行，而从用户的角度来看，就像是对象上平白增加了行为。

4.5 Autoproxing

自动代理可以不用为每一个目标对象手动定义代理对象，使用自动代理，您可以通过 Bean 名称或是 Pointcut 对比，自动为符合对比条件的目标对象建立代理对象。

4.5.1 BeanNameAutoProxyCreator

如果要为目标对象提供 Advice，则必须为它们建立代理对象，在应用程序规模大时，如果要提供 Advice 的目标对象很多，一个一个为它们建立代理对象会是件麻烦的事，为此，Spring 为一些情况提供了自动代理。

您可以为目标对象取好适当的 Bean 名称，例如为某些服务对象取名为 xxxService，这么一来，可以使用 org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator 来为这些 Bean 设置自动代理，例如 DelegatingIntroductionDemo 中就可以改用 BeanNameAutoProxyCreator 来建立自动代理，只要改一下 Bean 定义文件就可以了：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someService"
          class="onlyfun.caterpillar.Some"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="lockIntroduction"
          class="onlyfun.caterpillar.LockIntroduction"/>

    <bean id="lockAdvisor"
          class="org.springframework.aop.
                 → support.DefaultIntroductionAdvisor">
          <constructor-arg ref="lockIntroduction"/>
          <constructor-arg ref="onlyfun.caterpillar.ILockable"/>
    </bean>

    <bean id="introductionProxyCreator"
          class="org.springframework.aop.framework.
                 → autoproxy.BeanNameAutoProxyCreator">
        <property name="beanNames">
            <list>
                <value>*Service</value>
            </list>
        </property>
        <property name="interceptorNames" value="lockAdvisor"/>
    </bean>
</beans>
```

执行的结果不变，而这样的设置，如果打算让某个目标对象应用 Advice 时，只要将其名称定为 xxxService 就可以了，Spring 会自动建立代理对象。

4.5.2 DefaultAdvisorAutoProxyCreator

Spring 所提供的自动代理建立者（Autoproxy creator）还有 org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator，只要在定义文件中加上 DefaultAdvisorAutoProxyCreator 的设置，在 Bean 定义文件被读取完之后，DefaultAdvisorAutoProxyCreator 会自动搜寻所有的 Advisor，并自动将 Advisor 应用至符合 Pointcuts 的目标对象上。

例如可以将 BeanNameAutoProxyCreator 中介绍的 Bean 定义文件更改为如下例所示，其余的什么都不改，执行的结果也是相同的：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="someService"
         class="onlyfun.caterpillar.Some"/>

    <bean id="lockIntroduction"
         class="onlyfun.caterpillar.LockIntroduction"/>

    <bean id="lockAdvisor"
         class="org.springframework.aop.
                → support.DefaultIntroductionAdvisor">
         <constructor-arg ref="lockIntroduction"/>
         <constructor-arg value="onlyfun.caterpillar.ILockable"/>
    </bean>

     <bean id="autoProxyCreator"
          class="org.springframework.aop.framework.
                → autoproxy.DefaultAdvisorAutoProxyCreator"/>
</beans>
```

例如将 4.3.2 节中介绍 RegExpMethodPointcutAdvisor 的 RegexpDemo 项目中的 Bean 定义文件修改如下，则执行结果不变：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

<bean id="logBeforeAdvice"
         class="onlyfun.caterpillar.LogBeforeAdvice"/>

    <bean id="regExpAdvisor"
         class="org.springframework.aop.
                → support.RegexpMethodPointcutAdvisor">
       <property name="pattern" value=".*Newbie"/>
       <property name="advice" ref="logBeforeAdvice"/>
    </bean>

    <bean id="helloSpeaker"
         class="onlyfun.caterpillar.HelloSpeaker"/>

    <bean id="autoProxyCreator"
          class="org.springframework.aop.framework.
                → autoproxy.DefaultAdvisorAutoProxyCreator"/>
</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

建立自动代理是 Spring 在编写设置文件时的一个方便工具，但相对的，Bean 定义文件容易有些不清不楚的设置，因为不像直接自行在定义文件上编写来得清楚，并且您要小心地定义 Pointcut，以免 Advice 应用到不该应用的目标对象上。

4.6　Spring 2.0 的 AOP 支持

在前节中可以看到，AOP 的 Advice 必须实现特定接口，而配置设置依赖于 XML 的繁琐设置。在 Spring 2.0 之后，对于 AOP 功能的实现与设置新增了两种方式：一种是基于 XML Schema 的设置，一种是基于 Annotation 的支持。两种方式对于 AOP 在使用上的简化都有极大的帮助，这一节将介绍 Spring 2.0 新的 AOP 支持。

4.6.1　Before Advice：基于 XML Schema

在这里以实际的例子来说明 Spring 2.0 在 AOP 的实现与设置上的改变，以改写 4.2.1 节 Before Advice 的 BeforeAdviceDemo 项目为例，LogBeforeAdvice 现在不需要实现 MethodBeforeAdvice，改写后的程序代码如下所示：

BeforeAdviceDemo2　　LogBeforeAdvice.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;
import org.aspectj.lang.JoinPoint;

public class LogBeforeAdvice {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    public void before(JoinPoint jointPoint) {
        logger.log(Level.INFO,
            "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }
}
```

在程序代码中，before()方法为任意取的名称，它可以接受 JointPoint 实例，也可以省略。可以使用 JointPoint 的 getTarget()方法取得目标对象，使用 getArgs()取得调用的方法参数、使用 getSignature()方法取得 Pointcut 签名等信息。事实上在 Spring 2.0 中，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

任何一个 Advice 的方法实现都可以自行决定是否接受 JointPoint 实例。

为了使用 Spring 2.0 基于 XML Schema 的 AOP 设置方式，您必须在 XML 设置档的开头加入 aop 的名称空间声明：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
    ...
</beans>
```

加入 aop 的名称空间声明之后，就可以使用 Spring 2.0 新的<aop>标签，在这里先将改写后的 *beans-config.xml* 列出，稍后再加以解释每个设置项目的意义：

BeforeAdviceDemo2 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="logBeforeAdvice"
          class="onlyfun.caterpillar.LogBeforeAdvice"/>

    <bean id="helloSpeaker"
          class="onlyfun.caterpillar.HelloSpeaker"/>

    <aop:config>
        <aop:aspect id="logging" ref="logBeforeAdvice">
         <aop:before
          pointcut="execution(* onlyfun.caterpillar.IHello.*(..))"
          method="before"/>
   </aop:aspect>
    </aop:config>

</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

首先可以看到，不再需要明确声明使用 ProxyFactoryBean，所有的 AOP 配置是在<aop:config>标签中设置的；<aop:aspect>标签定义 Aspect 的实现，也就是 Advice 的实例。

<aop:before>标签表示设置的 Advice 将作为 Before Advice，"pointcut"属性定义 Pointcut 表示式，以上定义了 3 个部分：传回值、方法名称与参数类型。传回值设置“*”，表示任何传回值类型都符合，方法名称设置 onlyfun.caterpillar.IHello.*表示 IHello 接口所声明的任意方法都符合，参数类型设置“(..)”表示任何参数类型声明都符合（稍后还会详细介绍 Pointcut 表示式的编写）。"method"属性设置 Advice 上要调用的方法，在这里设置调用 LogBeforeAdvice 的 before()方法。

设置好 Bean 配置文件之后，接着改写一下 SpringAOPDemo 类，设置文件中没有明确设置代理对象了，所以只要直接取得"helloSpeaker"实例，Spring 就会自动按配置设置建立代理对象，并在调用对应方法时应用 Advice，SpringAOPDemo 改写如下：

BeforeAdviceDemo2 SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
    public static void main(String[] args) {
        ApplicationContext context =
               new ClassPathXmlApplicationContext(
                      "beans-config.xml");
        IHello helloSpeaker =
            (IHello) context.getBean("helloSpeaker");
        helloSpeaker.hello("Justin");
    }
}
```

执行的结果如图 4.15 所示。

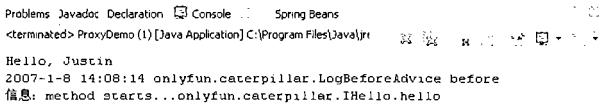


图 4.15 BeforeAdviceDemo2 项目的执行结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果需要重用 Pointcut 定义，可以使用<aop:pointcut>标签，例如这个范例的<aop:config>中也可以改写为以下的内容：

```
<aop:config>
    <aop:pointcut id="logHello"
      expression="execution(* onlyfun.caterpillar.IHello.*(..))"/>
    <aop:aspect id="logging" ref="logBeforeAdvice">
     <aop:before
       pointcut-ref="logHello"
       method="before"/>
 </aop:aspect>
  </aop:config>
```

良葛格的话匣子<<<

*相关的.jar 文件必须加入 Classpath，您可以直接使用 spring.jar，另外还需要 aspectjweaver.jar、asm- *.jar 与 asm-commons-*.jar。*

4.6.2 Before Advice：基于 Annotation

Spring 2.0 中还可以基于 Annotation 来设置 AOP 的 Advice，在 XML 的设置上可以更加简化，这里改写 BeforeAdviceDemo2 项目，将当中相关的设置改为使用 Annotation 的方式，首先可以改写 LogBeforeAdvice 如下：

BeforeAdviceDemo3　　LogBeforeAdvice.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class LogBeforeAdvice {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    @Before("execution(* onlyfun.caterpillar.IHello.*(..))" )
    public void before(JoinPoint jointPoint) {
        logger.log(Level.INFO,
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }
}
```

要将一个类设置为 Aspect，只要使用@Aspect 在类上标示它是 Aspect 的实现即可；@Before 表示该方法在应用为 Before Advice 时调用，当中直接编写 Pointcut 表示式，同样的，before()方法中的 JointPoint 参数可有可无，主要目的是让您取得一些与 JointPoint 相关的信息。

接着在 XML 配置文件中，使用<aop:aspectj-autoproxy/>标签启用@Aspect 的 Annotation 支持，*beans-config.xml* 改写如下：

BeforeAdviceDemo3 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="logBeforeAdvice"
          class="onlyfun.caterpillar.LogBeforeAdvice"/>

    <bean id="helloSpeaker"
          class="onlyfun.caterpillar.HelloSpeaker"/>

    <aop:aspectj-autoproxy/>

</beans>
```

可以看到，基于 Annotation 的 AOP 设置方式，在 XML 上几乎不用设置，只要实例化 Advice 与您的目标对象，接着一切就让 Spring 自动取得 Annotation 信息、进行代理对象建立吧，无须再作任何的设置。这个范例的执行结果与上一个范例是相同的，可以参考图 4.12 的执行结果画面。

良葛格的话匣子<<<

Spring 中 3 种 AOP 的实现与设置方式，哪个比较好？在 Spring 参考文件的 6.4. Choosing which AOP declaration style to use 给了些建议，您可以参考看看。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4.6.3 Spring 2.0 的 Pointcut 定义

在 Spring 2.0 中要声明 Pointcut，主要包括两个部分：Pointcut 表示式（expression）与 Pointcut 签名（signature）。首先来看 Pointcut 表示式，在之前的范例就已经使用过的是 executeion 表示式，例如：

```
execution(* onlyfun.caterpillar.IHello.*(..))
```

在 Spring 中，execution 表示式会是最常使用的 Pointcut 表示式，它的语法组成格式如下所示：

```
execution(modifiers-pattern?
          ret-type-pattern
          declaring-type-pattern?
          name-pattern(param-pattern)
        throws-pattern?)
```

它们分别表示存取修饰匹配（modifiers-pattern?）、传回值类型匹配（ret-type-pattern）、类类型匹配（declaring-type-pattern?）、方法名称匹配（参数类型匹配）（name-pattern(param-pattern)）、异常类型匹配（throws-pattern?），有问号的部分，表示可以省略不声明。

大部分情况下，传回值类型匹配会使用“*”，表示所有传回值类型都符合，也可以设置完整类型名称（fully-qualified type name）。方法名称匹配也可以使用“*”。参数类型匹配，用`()`表示没有参数，`(type)`表示带有一个类型为 `type` 的参数；`(*)`表示带有一个参数，而该参数可以是任意类型；也可以设置如`(*, String)`，表示有两个参数，第一个参数可以是任意类型，而第二个参数是 `String`；`(..)`表示零个或多个参数。

接下来举几个 execution 表示式的实际例子：

- execution(public * *(..))

 符合任何的公开方法。

- execution(* hello*(..))

 符合任何以 hello 开头的方法。

- execution(* onlyfun.caterpillar.IHello.*(..))

 符合 `IHello` 接口所声明的任何方法（之前的范例就是这么声明）。

- execution(* onlyfun.caterpillar.service.*.*(..))

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

符合 onlyfun.caterpillar.service 包（package）下声明的任何方法。

- execution(* onlyfun.caterpillar.service..*.*(..))

 符合 onlyfun.caterpillar.service 包或子包下声明的任何方法。

 如果只是要定义符合某些类型，则可以使用 within 表示式，直接以实例来示范：

- within(onlyfun.caterpillar.service.*)

 符合 onlyfun.caterpillar.service 包下声明的任何方法。

- within(onlyfun.caterpillar.service..*)

 符合 onlyfun.caterpillar.service 包或子包下声明的任何方法。

另外还有 this、target、args 等 Pointcut 表示式的使用，若有兴趣，可以参考 Spring 参考文件，基于篇幅，这里并不一一加以说明。

在 Spring 2.0 中，结合@Pointcut 的 Annotation 定义方式，可以让 Pointcut 在定义上更加容易，定义包括两个部分：Pointcut 表示式与 Pointcut 签名（signature）。例如以下定义一个符合 hello 开头方法的 Pointcut：

```
@Pointcut("execution(* hello*(..))")  // Pointcut 表示式
private void anyHello();  // Pointcut 签名
```

若要使用所定义的 Pointcut，则可以指定 Pointcut 签名，例如：

```
@Before("anyHello()" )
```

以上的例子，相当于直接在@Before 中定义以下的 Pointcut 表示式：

```
@Before("execution(* hello*(..))" )
```

Pointcut 定义时，还可以使用&&、||、!运算，例如在以下的定义中，anyHelloWelcome()签名将符合任何以 hello 与 welcome 开头的方法：

```
@Pointcut("execution(* hello*(..))")
private void anyHello();

@Pointcut("execution(* welcome*(..))")
private void anyWelcome();

@Pointcut("anyHello() && anyWelcome()")
private void anyHelloWelcome();
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

还可以将共用的一些 Pointcut 组织起来，以供整个应用程序使用，例如可以定义一个 CommonPointcut：

```
package onlyfun.caterpillar;

public class CommonPointcut {
    @Pointcut("execution(* hello*(..))")
    private void anyHello();

    @Pointcut("execution(* welcome*(..))")
    private void anyWelcome();

    @Pointcut("execution(* onlyfun.caterpillar.service..*.*(..))")
    private void service();

    @Pointcut("anyHello() && anyWelcome()")
    private void anyHelloWelcome();

    ...
}
```

要使用所定义的 Pointcut 时，可以指定完整类名称加上 Pointcut 签名，例如：

```
...
@Aspect
public class LogBeforeAdvice {
    ...
    @Before("onlyfun.caterpillar.CommonPointcut.service()")
    public void before() {
        ...
    }
}
```

或者是在使用 aop 标签时，指定其"pointcut"属性，例如：

```
<aop:before pointcut="onlyfun.caterpillar.CommonPointcut.service()"
            method="before"/>
```

4.6.4 After Returning Advice：基于 XML Schema

在 Spring 2.0 中，可以使用<aop:after-returning>标签来定义 After Returning Advice，而不用实现 AfterReturningAdvice 接口，例如可以改写 4.2.2 节 After Advice 的 AfterAdviceDemo 项目，在当时是将 Before Advice 与 After Advice 分开编写为两个类，这里则将之合并为一个类，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

AfterAdviceDemo2 LogAspect.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.aspectj.lang.JoinPoint;

public class LogAspect {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    public void before(JoinPoint jointPoint) {
        logger.log(Level.INFO, "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }

    public void afterReturning(JoinPoint jointPoint) {
        logger.log(Level.INFO, "method ends..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }
}
```

LogAspect 中，before()方法将作为 Before Advice 来调用，而 afterReturing()方法将作为 After Returning Advice 来调用，After Returning Advice 还可以设置目标方法调用的传回值，这在稍后介绍基于 Annotation 的 After Returning Advice 时还会示范。接着改写 Bean 定义文件的内容，使用<aop:after-returning>标签：

AfterAdviceDemo2 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="logAspect"
          class="onlyfun.caterpillar.LogAspect"/>

    <bean id="helloSpeaker"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        class="onlyfun.caterpillar.HelloSpeaker"/>

    <aop:config>
        <aop:aspect id="logging" ref="logAspect">
            <aop:pointcut id="logHello" expression=
                "execution(* onlyfun.caterpillar.IHello.*(..))"/>

        <aop:before pointcut-ref="logHello" method="before"/>

        <aop:after-returning
            pointcut-ref="logHello"
            method="afterReturning"/>
    </aop:aspect>
    </aop:config>

</beans>
```

<aop:after-returning>标签上的属性作用，与<aop:before>标签是类似的，请参考先前介绍<aop:before>标签时的说明。配置文件中没有明确设置代理对象了，所以只要取得了"helloSpeaker"实例，Spring 就会自动依配置设置建立代理对象，并在调用对应方法时应用 Advice，SpringAOPDemo 的改写与先前 BeforeAdviceDemo2 项目是相同的，这里不再列出。以下是执行时的参考画面，如图 4.16 所示。

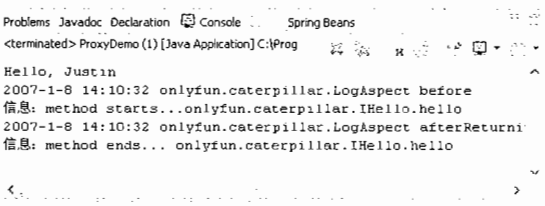


图 4.16 AfterAdviceDemo2 项目的执行结果

4.6.5 After Returning Advice：基于 Annotation

如果要使用 Spring 2.0 基于 Annotation 的 After Returning Advice 定义方式，则要使用@AfterReturing，以下改写 AfterAdviceDemo2 项目的 LogAspect 类：

AfterAdviceDemo3 LogAspect.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class LogAspect {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    @Pointcut("execution(* onlyfun.caterpillar.IHello.*(..))")
    private void logging() {}

    @Before("logging()" )
    public void before(JoinPoint jointPoint) {
        logger.log(Level.INFO, "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }

    @AfterReturning(pointcut="logging()", returning="retVal")
    public void afterReturning(JoinPoint jointPoint, Object retVal) {
        logger.log(Level.INFO, "method ends..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }
}
```

在这个 LogAspect 中，还示范了如何接受目标方法执行完毕后的传回值，也就是设置"returning"属性，使用 retVal 名称来接受传回值。也就是说，有机会修改其所引用的对象。接着再修改一下 *beans-config.xml*：

AfterAdviceDemo3　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="logAdvice"
          class="onlyfun.caterpillar.LogAspect"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="helloSpeaker"
          class="onlyfun.caterpillar.HelloSpeaker"/>

    <aop:aspectj-autoproxy/>

</beans>
```

同样的，只要使用<aop:aspect-autoproxy>标签，并设置 Aspect 的实例即可。执行的结果与上一个范例结果是相同的。

4.6.6 After Throwing Advice：基于 XML Schema

After Throwing Advice 的作用即 4.2.4 节所介绍的 Throw Advice 的作用，可以使用<aop:after-throwing>来定义，而不用实现 ThrowsAdvice 接口，例如改写先前 ThrowAdviceDemo 项目的内容，以下的 LogAspect 还结合了 Before Advice、After Returning Advice：

ThrowAdviceDemo2　　LogAspect.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.aspectj.lang.JoinPoint;

public class LogAspect {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    public void before(JoinPoint jointPoint) {
        logger.log(Level.INFO, "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }

    public void afterReturning(JoinPoint jointPoint) {
        logger.log(Level.INFO, "method ends..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }

    public void afterThrowing(JoinPoint jointPoint,
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                        Throwable throwable) {
        logger.log(Level.INFO, "Logging that a " + throwable +
            "\nException was thrown in..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }
}
```

afterThrowing()方法将在发生 Throwable 或其子类异常时被调用，在这里，您也可以设置其他子类类型，Spring 将自动更新所引发的异常类型，决定是否调用 afterThrowing()方法，"throwable"参数名必须在<aop:after-throwing>的"throwing"属性当中设置，如下所示：

ThrowAdviceDemo3　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="logAspect"
         class="onlyfun.caterpillar.LogAspect"/>

    <bean id="helloSpeaker"
         class="onlyfun.caterpillar.HelloSpeaker"/>

    <aop:config>
        <aop:aspect id="logging" ref="logAspect">
            <aop:pointcut id="logHello" expression=
              "execution(* onlyfun.caterpillar.IHello.*(..))"/>

            <aop:before pointcut-ref="logHello" method="before"/>

            <aop:after-returning
                 pointcut-ref="logHello"
                 method="afterReturning"/>

            <aop:after-throwing
                 pointcut-ref="logHello"
                 throwing="throwable"
                 method="afterThrowing"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        </aop:aspect>
    </aop:config>

</beans>
```

而原本 ThrowAdviceDemo 项目的 SpringAOPDemo 类则修改如下：

ThrowAdviceDemo3 SpringAOPDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {
    public static void main(String[] args) {
        ApplicationContext context =
               new ClassPathXmlApplicationContext(
                     "beans-config.xml");
        IHello helloSpeaker =
            (IHello) context.getBean("helloSpeaker");

        try {
            helloSpeaker.hello("Justin");
        }
        catch(Throwable throwable) {
            // 应用程序的异常处理
            System.err.println(throwable);
        }
    }
}
```

执行的结果如图 4.17 所示。

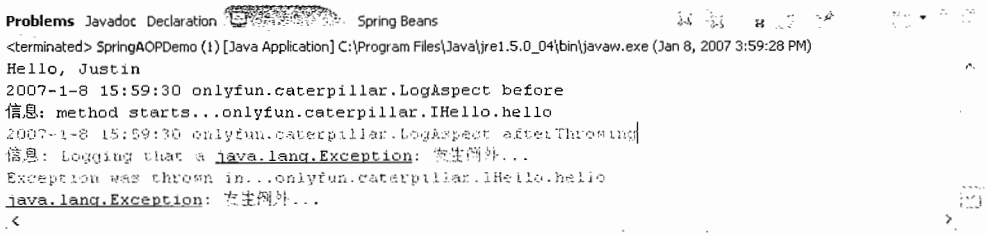


图 4.17 ThrowAdviceDemo2 项目的执行结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

从执行结果中也可以看到，由于方法执行过程中发生异常了，方法并未正确执行完毕，所以 After Returning Advice 不会执行。

4.6.7　After Throwing Advice：基于 Annotation

基于 Annotation 的 After Throwing Advice 方式要使用@AfterThrowing，这里直接将之前 ThrowAdviceDemo2 项目的内容修改为基于 Annotation 的方式，首先是 LogAspect 类的修改：

ThrowAdviceDemo3　　LogAspect.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class LogAspect {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    @Pointcut("execution(* onlyfun.caterpillar.IHello.*(..))")
    private void logging() {}

    @Before("logging()" )
    public void before(JoinPoint jointPoint) {
        logger.log(Level.INFO, "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }

    @AfterReturning(pointcut="logging()", returning="retVal")
    public void afterReturning(JoinPoint jointPoint, Object retVal) {
        logger.log(Level.INFO, "method ends..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    @AfterThrowing(pointcut="logging()", throwing="throwable")
    public void afterThrowing(JoinPoint jointPoint,
                              Throwable throwable) {
        logger.log(Level.INFO, "Logging that a " + throwable +
            "\nException was thrown in..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());
    }
}
```

接着是 *beans-config.xml* 的修改：

ThrowAdviceDemo3 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="logAdvice"
         class="onlyfun.caterpillar.LogAspect"/>

    <bean id="helloSpeaker"
         class="onlyfun.caterpillar.HelloSpeaker"/>

    <aop:aspectj-autoproxy/>

</beans>
```

由于这里将所有的 Advice 编写在 `LogAspect` 类中，所以事实上 *beans-config.xml* 与先前在 AfterAdviceDemo3 项目中是相同的设置，这里再次列出只是为了强调。执行的结果则与 ThrowAdviceDemo2 项目是相同的。

4.6.8 Around Advice：基于 XML Schema

Spring 2.0 中要定义 Around Advice，不用实现 `MethodInterceptor` 接口，Advice 的方法名称可以任意，但必须传回对象，必须定义一个接受 `ProceedingJointPoint` 实例的参数，例如：

```
public Object invoke(ProceedingJoinPoint jointPoint) throws Throwable {
    ...
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

就如4.2.3节Around Advice中所介绍的MethodInvocation一样，ProceedingJointPoint也有一个 proceed()，在真正调用 proceed()方法之前，目标对象的方法不会被执行，而proceed()方法执行完毕后会传回一个对象，它表示目标对象的方法执行完毕后的传回值，在 Around Advice 方法执行完毕后可以传回这个对象，甚至传回另一个对象。

以下将先前 AfterAdviceDemo2 项目的内容修改为使用 Around Advice，首先是 LogAspect 类：

AroundAdviceDemo2 LogAspect.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;
import org.aspectj.lang.ProceedingJoinPoint;

public class LogAspect {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    public Object invoke(
                ProceedingJoinPoint jointPoint) throws Throwable {
        logger.log(Level.INFO, "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());

        Object retVal = jointPoint.proceed();

        logger.log(Level.INFO, "method ends..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());

        return retVal;
    }
}
```

beans-config.xml 修改如下：

AroundAdviceDemo2 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <bean id="logAspect"
         class="onlyfun.caterpillar.LogAspect"/>

    <bean id="helloSpeaker"
         class="onlyfun.caterpillar.HelloSpeaker"/>

    <aop:config>
        <aop:aspect id="logging" ref="logAspect">
          <aop:around pointcut=
            "execution(* onlyfun.caterpillar.IHello.*(..))"
            method="invoke"/>
  </aop:aspect>
    </aop:config>

</beans>
```

执行结果与 AfterAdviceDemo2 项目是相同的，请参考先前 AfterAdviceDemo2 项目的执行画面。

4.6.9 Around Advice：基于 Annotation

Around Advice 是基于 Annotation 的设置方式，所以它要使用@Around，例如可以直接修改 AfterAdviceDemo3 项目的 LogAspect，如下所示。

AroundAdviceDemo3 — LogAspect.java

```
package onlyfun.caterpillar;

import java.util.logging.Level;
import java.util.logging.Logger;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class LogAspect {
    private Logger logger =
            Logger.getLogger(this.getClass().getName());

    @Around("execution(* onlyfun.caterpillar.IHello.*(..))")
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public Object invoke(
            ProceedingJoinPoint jointPoint) throws Throwable {
        logger.log(Level.INFO, "method starts..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());

        Object retVal = jointPoint.proceed();

        logger.log(Level.INFO, "method ends..." +
            jointPoint.getSignature().getDeclaringTypeName() +
            "." + jointPoint.getSignature().getName());

        return retVal;
    }
}
```

在 Spring 进入 2.0 版本之后，于 AOP 这方面所增加的功能也是相当地多，由于书籍篇幅有限，无法一一为您作介绍，建议可以查阅 Spring 参考手册，以获得更多高级内容的说明。

4.7　接下来的主题

Spring 中所有子框架的基础就是 Spring 的 IoC 容器与 Spring 的 AOP 支持，在第 3 章与第 4 章中已经介绍完毕，接下来就可以开始探索 Spring 中所提供的子框架或 API，首先要来认识的是 Spring 在数据库存取方面的支持，包括了对 JDBC 的封装、事务管理等议题。



图 4.18　Spring AOP 主要的原理就是代理

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第

JDBC、事务支持

章

对于数据库存取的支持，Spring 提供了 DAO 框架，您可以不用接触底层数据库的技术细节。Spring 对数据库存取时的一些异常加以重新封装，您可以选择是否处理特定的异常。在 JDBC API 的使用上，Spring 也提供了 `JdbcTemplate` 等类以简化 JDBC 在 API 上的操作使用。关于事务的处理方面，Spring 提供了编程式事务管理（Programmatic transaction management）与声明式事务管理（Declarative transaction management）。

总而言之，结合 Spring IoC 容器与 AOP 框架所提供的功能，可以简化 JDBC 的运用方式，并轻易获得事务等相关服务的功能。在这个章节中，将以一些实际的例子来介绍 Spring 于 JDBC 上的存取以及事务上的一些支持。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.1　Spring 持久层入门

Spring 提供了 DAO 框架，让程序员在开发应用程序时无须耦合于特定数据库技术，这节先来简介 Spring 的 DAO 设计，并以实际的例子来入门 Spring 持久层。

5.1.1　Spring 的 DAO 支持

Spring 的 DAO 框架让您在进行数据库存取时，无须接触到所使用的特定数据库技术细节，DAO 的全名为 Data Access Object，在应用程序中，需要使用到数据存取时，是通过一个数据存取接口来操作的，而实际上进行数据库存取的对象要实现该接口，并在规范的方法之中实现存取时的相关细节。

举个 DAO 的例子，假设应用程序中有个 User 对象，在进行数据库存取时（例如 select、insert、update、delete），应用程序不应依赖于一个实际的类实现，而可以让它依赖一个接口，例如一个 IUserDAO 接口：

```
package onlyfun.caterpillar;

public interface IUserDAO {
    public void insert(User user);
    public User find(Integer id);
    public void update(User user);
    public void delete(User user);
}
```

实际上进行数据库存取的类可以实现 IUserDAO 接口，例如定义一个简单的 UserDAO 类：

```
package onlyfun.caterpillar;
...
public class UserDAO implements IUserDAO {
    private DataSource dataSource;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void insert(User user) {
       String name = user.getName();
       int age = user.getAge().intValue();
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        Connection conn = null;
        PreparedStatement stmt = null;
        try {
            conn = dataSource.getConnection();
            stmt = conn.prepareStatement (
                "INSERT INTO user (name,age) VALUES(?,?)");
            stmt.setString(1, name);
            stmt.setInt(2, age);
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally {
            if(stmt != null) {
                try {
                    stmt.close();
                }
                catch(SQLException e) {
                    e.printStackTrace();
                }
            }
            if(conn != null) {
                try {
                    conn.close();
                }
                catch(SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public User find(Integer id) {
        ...
        return user;
    }

    public void update(User user) {
        ...
    }

    public void delete(User user) {
        ...
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

您的应用程序主流程在进行数据存取时，可以使用 IUserDAO 来声明操作接口，例如：

```
...
User user = new User();
user.setName("caterpillar");
user.setAge(new Integer(30));

IUserDAO userDao = getUserDAO();
userDao.insert(user);
...
```

由于依赖于接口，所以可以随时替换 IUserDAO 的实现类，而在 IUserDAO 接口声明的操作方法上，并没有任何底层数据库存取的技术细节，Spring 的 DAO 框架正是基于这样的基本原理，将应用程序与底层存取技术隔离开来，如图 5.1 所示。

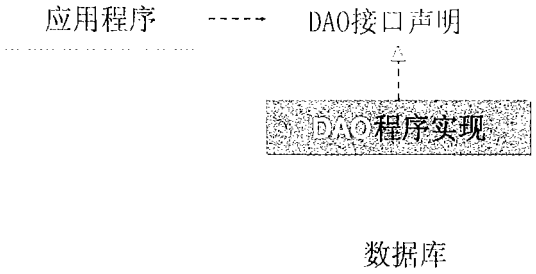


图 5.1　DAO 运作示意图

数据存取接口上只有与特定数据库存取技术无关的方法（例如 update、insert、delete 等），设计上依赖于接口，程序也易于测试，不让应用程序受限于只能使用某一数据库技术。

在之前的示范程序中，对于实际的数据库存取流程来说，有几个步骤是固定的，例如取得 DataSource、取得 Connection、处理异常等，对于不同的数据库技术，这些步骤大致上是相同的，只有少部分流程不同，就设计上而言，可以使用 Template-Callback 模式，将固定的流程编写于 Template 类之中，而对于不同的一些细节步骤，则委托 Callback 对象来完成，例如以下的程序片段中，非粗体字部分是固定的流程，对于其中有差异的部分，可以实现个别的 Callback 对象来执行或传回适当的对象：

```
...
Connection conn = null;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
PreparedStatement stmt = null;
try {
    conn = dataSource.getConnection();
    stmt = callback.createPreparedStatement(conn);
    stmt.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
finally {
    if(stmtCallback != null) {
        try {
            stmt.close();
        }
        catch(SQLException e) {
            e.printStackTrace();
        }
    }
    if(conn != null) {
        try {
            conn.close();
        }
        catch(SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Spring 运用了 Template-Callback 模式，将固定的流程编写于 Template 类之中（例如 JdbcTemplate、HibernateTemplate 类），而对于不同的一些细节步骤，则委托特定 DAO 支持对象来处理（可以自定义或由 Spring 自动产生），如图 5.2 所示。

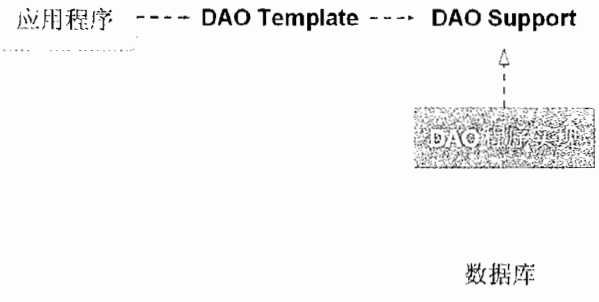


图 5.2 DAO Template 与 DAO Support

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在异常处理方面，Spring 也提供与特定技术无关的异常处理体系，让应用程序不会因处理特定异常（像是 SQLException、HibernateException）而耦合于某种数据库或持久层技术。

首先来了解异常处理，Java 的异常有 Checked exception 与 Unchecked exception 两种。Checked exception 是编译时期在语法上必须处理的异常，因为这些异常通常是可以预期发生的，编译器要求一定要处理，因此必须在语法上以 try...catch 加以处理；Unckecked exception 则是执行时期异常（Runtime exception），在异常继承架构上是 java.lang.RuntimeException 类的子类，通常是由于程序逻辑上的错误，在执行时期所引发的异常，异常真的发生时，可以选择使用 try...catch 来作处理，或是让异常直接丢出至应用程序最上层处理（例如用户图形接口）。

Checked exception 的立意本来是好的，当这类异常发生时，希望的是程序设计人员可以加以处理至程序能回复正常运作，然而有时候，出现了 Checked exception 往往无力回复至正常运作，当这类异常在底层的数据库存取时发生（例如无法取得联机），最好的处理方式就是不处理，让异常传播至上层应用程序，由上层应用程序捕捉以显示相关消息，让用户得知问题出在哪边，而不是在底层的数据库存取程序中，作一些无能为力的处理（例如记录下无法联机的消息），然而面对 Checked exception，由于编译器要求一定要处理，程序设计人员只好无可奈何的编写一些异常处理语法来处理掉这些异常。

直接使用 throws 在方法上声明异常，让异常可以向上层传播也不是一个好的主意，另一方面，有些程序或框架会自行继承相关的异常类，包括一些相关的异常消息，它们也会在定义接口时，于方法上声明 throws 某些类型的异常，然而如果在这些方法中发生了某些不是方法上声明的异常（可能由于使用的底层技术不同而有这种情况，像是 JDBC 或是 Hibernate），就无法将之 throw，只能自行编写一些 try..catch 来暗自处理掉，如果想要让这些异常丢出至上层，就要更多道的手续。

Spring 的 DAO 框架并不丢出与数据库技术相关的异常，Spring 所有的异常都是 org.springframework.dao.DataAccessException 的子类，一个与数据库技术无关的通用异常，而且 DataAccessException 是 RuntimeException 的子类，也就是说它是属于 Unchecked exception，您不用被强迫使用 try...catch 来处理异常，而可以自己选择要不要处理，在不处理异常的情况下，也可以很简单地将异常传播至上层的应用程序。

对于 JDBC 存取，Spring 将 SQLException 等转化为自己的 DAO 异常对象，org.springframework.dao 包下提供一致性的异常处理层次，DataAccessException 是这个层次的基础类，它继承自 org.springframework.core 包的 NestedRuntimeException，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

而 NestedRuntimeException 继承自 RuntimeException，对于一些异常，您可以选择处理它或者忽略它，由最上层的应用程序或是最后由 JVM 来处理。

如果要处理特定的异常，Spring 也将异常作好了分类，例如数据库连结错误时会丢出的 DataAccessResourceFailureException、SQL 语法错误时的 BadSqlGrammarException 等，您可以针对想处理的异常加以捕捉。可以看看 Spring 参考手册中的 DAO support 章节，当中有一些对异常处理的说明，也有个 DataAccessException 的类继承图。

5.1.2 DataSource 注入

对于不同的数据连接来源需求，Spring 提供了 javax.sql.DataSource 注入，要更换数据来源只要在 Bean 定义文件中修改配置，而不用修改任何一行程序。

因为不同的系统，应用程序可能使用不同的数据来源，例如纯粹的使用 JDBC、通过连接池、或是通过 JNDI 等，数据的来源更多是底层的行为，这些行为不应影响到上层的业务逻辑，为此，可以在需要取得连接来源的 Bean 对象上保留一个 Datasource 注入的操作接口，Spring 提供 org.springframework.jdbc.datasource.DriverManagerDataSource 来取得它的实例，DriverManagerDataSource 实现了 javax.sql.DataSource，您可以在 Bean 定义文件中这么编写：

```
...
<beans...>
    <bean id="dataSource"
         class="org.springframework.jdbc.
                  → datasource.DriverManagerDataSource">
        <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
        <property name="url"
             value="jdbc:mysql://localhost:3306/demo"/>
        <property name="username" value="caterpillar"/>
        <property name="password" value="123456"/>
    </bean>
    ...
</beans>
```

其中 "driverClassName"、"url"、"username"、"password" 4 个属性分别是用来设置 JDBC 驱动程序类、数据库 URL 协定、用户名称和密码的。

在这里实际使用一个程序来作为 DataSource 注入的示范，并在示范之前有关于 DAO 介绍的实现，假设您定义了一个 DAO 的操作接口，如下所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

DataSourceDemo　　IUserDAO.java

```
package onlyfun.caterpillar;

public interface IUserDAO {
    public void insert(User user);
    public User find(Integer id);
}
```

基于篇幅的限制，这里关于 DAO 的接口定义只介绍了 `insert()` 与 `find()` 两种方法。在这个接口中所使用到的 `User` 类则如下定义：

DataSourceDemo　　User.java

```
package onlyfun.caterpillar;

public class User {
    private Integer id;
    private String name;
    private Integer age;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接着可以定义一个 UserDAO 类，它实现了 IUserDAO 接口，是实际进行数据库存取服务的对象，如下所示：

DataSourceDemo UserDAO.java

```
package onlyfun.caterpillar;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;

public class UserDAO implements IUserDAO {
    private DataSource dataSource;

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void insert(User user) {
       String name = user.getName();
       int age = user.getAge().intValue();

       Connection conn = null;
       PreparedStatement stmt = null;
       try {
           conn = dataSource.getConnection();
           stmt = conn.prepareStatement (
               "INSERT INTO user (name,age) VALUES(?,?)");
           stmt.setString(1, name);
           stmt.setInt(2, age);
           stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally {
            if(stmt != null) {
                try {
                    stmt.close();
                }
                catch(SQLException e) {
                    e.printStackTrace();
                }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            }
            if(conn != null) {
                try {
                    conn.close();
                }
                catch(SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public User find(Integer id) {
        Connection conn = null;
        PreparedStatement stmt = null;

        try {
            conn = dataSource.getConnection();
            stmt = conn.prepareStatement(
                      "SELECT * FROM user WHERE id=?");
            stmt.setInt(1, id.intValue());

            ResultSet result = stmt.executeQuery();
            if(result.next()) {
                Integer i = new Integer(result.getInt(1));
                String name = result.getString(2);
                Integer age = new Integer(result.getInt(3));

                User user = new User();
                user.setId(i);
                user.setName(name);
                user.setAge(age);

                return user;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        finally {
            if(stmt != null) {
                try {
                    stmt.close();
                }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                catch(SQLException e) {
                    e.printStackTrace();
                }
            }
            if(conn != null) {
                try {
                    conn.close();
                }
                catch(SQLException e) {
                    e.printStackTrace();
                }
            }
        }

        return null;
    }
}
```

UserDAO 类上声明一个 setDataSource() 方法，可以让您注入 DataSource 的实例，可以在 Bean 定义文件中进行依赖注入的定义，如下所示：

DataSourceDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                 → datasource.DriverManagerDataSource">
        <property name="driverClassName"
                 value="com.mysql.jdbc.Driver"/>
        <property name="url"
                 value="jdbc:mysql://localhost:3306/demo"/>
        <property name="username" value="caterpillar"/>
        <property name="password" value="123456"/>
    </bean>

    <bean id="userDAO"
         class="onlyfun.caterpillar.UserDAO">
        <property name="dataSource" ref="dataSource"/>
    </bean>
</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以编写一个简单的测试程序来操作 UserDAO 的实例，看看是否能进行数据的储存与查询：

DataSourceDemo　　SpringDAODemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringDAODemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                    "beans-config.xml");

        User user = new User();

        user.setName("caterpillar");
        user.setAge(new Integer(30));

        IUserDAO userDAO =
            (IUserDAO) context.getBean("userDAO");

        userDAO.insert(user);

        user = userDAO.find(new Integer(1));

        System.out.println("name: " + user.getName());
    }
}
```

良葛格的话匣子<<<

在进行程序的测试之前，您要先开启数据库服务，并将 *beansconfig.xml* 中相关的驱动程序类、数据库 URL、用户名称与密码等修改为您的设置，在这里所使用的是 MySQL 数据库，而建立的 user 表格是使用以下的 SQL 建立的：

```
CREATE TABLE user (
    id INT(11) NOT NULL auto_increment PRIMARY KEY,
    name VARCHAR(100) NOT NULL default '',
    age INT
);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程序的执行结果会先在数据库的 user 表格中存入一笔数据，接着根据 id 值查询出先前所存入的数据，最后结果会显示 "name: caterpillar" 的文字。

良葛格的话匣子<<<

您需要 spring-core.jar、spring-beans.jar、spring-context.jar、spring-dao.jar 与 spring-jdbc.jar 这几个文件，如果您使用的是 spring.jar，当中已经包括了相关的 API，另外也需要相依赖的 commons-logging.jar，当然，为了使用 JDBC，您必须要有 JDBC 驱动程序的 jar 文件。

5.1.3 DataSource 置换

DriverManagerDataSource 并没有提供连接池的功能，只是用来作简单的单机连接测试，并不适合使用于真正的项目之中。您可以使用 DBCP 以获得连接池的功能。如果使用 Spring，则置换 DataSource 并不需要修改程序源代码，只要修改 Bean 定义文件就可以了，例如修改一下 DataSourceDemo 项目中的 *beans-config.xml* 如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
         class="org.apache.commons.dbcp.BasicDataSource"
         destroy-method="close">
        <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
        <property name="url"
                value="jdbc:mysql://localhost:3306/demo"/>
        <property name="username" value="caterpillar"/>
        <property name="password" value="123456"/>
    </bean>

    <bean id="userDAO"
         class="onlyfun.caterpillar.UserDAO">
        <property name="dataSource" ref="dataSource"/>
    </bean>
</beans>
```

现在所使用的是 org.apache.commons.dbcp.BasicDataSource 作为注入的 DataSource 实例，为了使用 DBCP 的功能，您需要在 *Classpath* 路径中设置 *commons-dbcp.jar*、

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

commons-pool.jar 与 *commonscollections.jar*，这些都可以在 Spring 所依赖版本中的 *lib* 目录下找到。

注意到在 dataSource 上设置了 "destroy-method" 属性，如此可以确保 BeanFactory 在关闭时也一并关闭 BasicDataSource。

如果 Servlet 容器提供了 JNDI（Java Naming and Directory Interface）的 DataSource，也可以简单地换上这个 DataSource：

```
...
<bean id="dataSource"
    class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name="jndiName" value="jdbc/demo"/>
</bean>
...
```

为了使用 org.springframework.indi.JndiObjectFactoryBean，您需要 *spring-context.jar*，"jndiName" 实际上要根据所设置的 JNDI 查询名称，例如在 Tomcat 中可以这么设置 *server.xml*：

```
...
<!-- 假设应用程序目录是 JSP -->
<Context path="/JSP" docBase="JSP">
   <!-- 使用数据库名称是 GUESTBOOK -->
   <Resource name="jdbc/demo" scope="Shareable"
           type="javax.sql.DataSource"/>
   <ResourceParams name="jdbc/demo">
      <parameter>
         <name>factory</name>
         <value>
            org.apache.commons.dbcp.BasicDataSourceFactory
         </value>
      </parameter>

      <!-- DBCP database connection settings -->
      <!-- JDBC URL -->
      <parameter>
        <name>url</name>
        <value>jdbc:mysql://localhost/demo</value>
      </parameter>
      <!-- JDBC 驱动程序 -->
      <parameter>
         <name>driverClassName</name>
         <value>com.mysql.jdbc.Driver</value>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            </parameter>
            <!-- 数据库用户与密码 -->
            <parameter>
                <name>username</name>
                <value>caterpillar</value>
            </parameter>
            <parameter>
                <name>password</name>
                <value>123456</value>
            </parameter>

            <!-- DBCP connection pooling options -->
            <!-- 等待 Connection 的时间，单位 ms，-1 表不限制 -->
            <parameter>
                <name>maxWait</name>
                <value>3000</value>
            </parameter>
            <!-- 连接池中最多可 idle 的 Connection 数，
                也就是最少的 Connection 数，0 表不限制 -->
            <parameter>
                <name>maxIdle</name>
                <value>10</value>
            </parameter>
            <!-- 连接池至多的 Connection 数，0 表示不限制 -->
            <parameter>
                <name>maxActive</name>
                <value>100</value>
            </parameter>
        </ResourceParams>
    </Context>
    ...
```

5.2 JDBC 支持

在使用 JDBC 时，总是要处理繁琐的细节，例如 Connection、Statement 的获得、SQLException 的处理、Connection、Statement 的关闭等问题，Spring 在 JDBC 的使用上提供了几个类，可以简化 JDBC 使用时的流程。

5.2.1 使用 JdbcTemplate

在前一节介绍的 DataSourceDemo 项目中，UserDAO 中直接使用 JDBC 来实现 insert()

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

与 find() 方法，当中要处理 Connection 的取得、Statement 的建立、异常的处理、Statement 的关闭、Connection 的关闭等，对于一个基本的 JDBC 存取，这些流程都是大同小异的，每一次都必须走这样的流程着实令人厌烦。

Spring 提供了 org.springframework.jdbc.core.JdbcTemplate 类，它被设计为线程安全(Thread-safe)，当中所提供的一些操作方法封装了类似以上的流程，例如 DataSourceDemo 项目中的 UserDAO 类可以简单地使用 JdbcTemplate 来改写，要建立 JdbcTemplate 的实例，必须要有一个 DataSource 对象作为构造时的对象：

```
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
```

来改写一下 DataSourceDemo 项目中 UserDAO 的内容实现，可以看到使用 JdbcTemplate 时，在程序的编写流程上会有什么样的改进：

JdbcTemplateDemo　　UserDAO.java

```
package onlyfun.caterpillar;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;

public class UserDAO implements IUserDAO {
    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public void insert(User user) {
       String name = user.getName();
       int age = user.getAge().intValue();

       jdbcTemplate.update("INSERT INTO user (name,age) "
             + "VALUES('" + name + "'," + age + ")");
    }

    public User find(Integer id) {
        List rows = jdbcTemplate.queryForList(
          "SELECT * FROM user WHERE id=" + id.intValue());

        Iterator it = rows.iterator();
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if(it.hasNext()) {
            Map userMap = (Map) it.next();

            Integer i = new Integer(userMap.get("id").toString());
            String name = userMap.get("name").toString();
            Integer age =
                new Integer(userMap.get("age").toString());

            User user = new User();
            user.setId(i);
            user.setName(name);
            user.setAge(age);

            return user;
        }

        return null;
    }
}
```

只要改写 UserDAO 就可以了，其他的程序与设置文件都不用变动，Spring 的 JdbcTemplate 一如其名，主要是通过 Template Method 模式来实现 JDBC 的流程封装处理。

良葛格的话匣子<<<

如果使用 Hibernate 之类的 ORM（Object-Relational Mapping）工具，则 find()方法中对象模型与关联式模型的转换还可以进一步的简化。

Spring 的 JDBC 封装等功能基本上可以独立于 Spring 来使用，除了 JdbcTemplate 之外，Spring 还提供了其他的 Template 类，像是对 Hibernate、JDO、iBatis 等的 Template 实现。在数据库的事务处理方面，Spring 提供了编程式与声明式的事务管理功能，简化了持久层程序的复杂度，并提供了更好的维护性。

5.2.2 执行与更新 JdbcTemplate

您可以使用 JdbcTemplate 的 execute() 方法执行 SQL 陈述，例如：

```
jdbcTemplate.execute(
    "CREATE TABLE USER (user_id integer, name varchar(100))");
```

如果是 UPDATE 或 INSERT，您可以使用 update() 方法，它有数个重载（Overload）版本，例如接受实现 org.springframework.jdbc.core.PreparedStatementCreator 接口的对象，PreparedStatementCreator 接口的定义如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
package org.springframework.jdbc.core;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public interface PreparedStatementCreator {
    PreparedStatement createPreparedStatement(Connection con)
        throws SQLException;
}
```

例如可以将 5.2.1 节的 JdbcTemplateDemo 项目中 UserDAO 的 insert() 方法改写如下：

```
...
public void insert(User user) {
   final String name = user.getName();
   final int age = user.getAge().intValue();

   jdbcTemplate.update(
     new PreparedStatementCreator() {
        public PreparedStatement createPreparedStatement(
              Connection con) throws SQLException {
           String sql =
              "INSERT INTO user (name,age) VALUES(?,?)";
           PreparedStatement ps =
                    con.prepareStatement(sql);
           ps.setString(1, name);
           ps.setInt(2, age);
           return ps;
        }
     });
}
...
```

在这个例子中，使用了 PreparedStatement 来准备 SQL，JdbcTemplate 实现了 Template-Callback 机制，update()方法中实现了 Template 流程，而 PreparedStatement-Creator 则实现了 Callback 对象，在执行 JDBC 的流程中，必要时会执行所定义的 callback 方法，也就是 createPreparedStatement()方法，传回一个 PreparedStatement 对象，以完成 update()方法中的 Template 流程。

与 PreparedStatementCreator 互补的接口是 org.springframework.jdbc.core.PreparedStatementSetter 接口：

```
package org.springframework.jdbc.core;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.sql.PreparedStatement;
import java.sql.SQLException;

public interface PreparedStatementSetter {
    void setValues(PreparedStatement ps) throws SQLException;
}
```

例如可以将 JdbcTemplateDemo 项目中 UserDAO 的 insert() 方法改写如下：

```
...
public void insert(User user) {
   final String name = user.getName();
   final int age = user.getAge().intValue();

   jdbcTemplate.update(
      "INSERT INTO user (name,age) VALUES(?,?)",
      new PreparedStatementSetter() {
           public void setValues(PreparedStatement ps)
                              throws SQLException {
              ps.setString(1, name);
              ps.setInt(2, age);
           }
        });
}
...
```

JdbcTemplate 会自动建立 PreparedStatementCreator 的实例，以提供传递给 setValues() 方法的 PreparedStatement 对象。

您也可以直接提供 SQL，就如 JdbcTemplateDemo 项目中所示范的 UserDAO 中的 insert() 方法一样：

```
...
public void insert(User user) {
   String name = user.getName();
   int age = user.getAge().intValue();

   jdbcTemplate.update("INSERT INTO user (name,age) "
         + "VALUES('" + name + "'," + age + ")");
}
...
```

在直接下 SQL 语句时，也可以使用 "?" 作为占位字符，并使用对象数组，把它作为参数传递给 JdbcTemplate 的 update() 方法，例如改写 JdbcTemplateDemo 项目中所示范的 UserDAO 中的 insert() 方法：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
...
public void insert(User user) {
   jdbcTemplate.update(
      "INSERT INTO user (name, age) VALUES(?,?)",
      new Object[] {user.getName(), user.getAge()});
}
...
```

JdbcTemplate 会自动建立 PreparedStatementCreator 与 PreparedStatementSetter 的实例，然而这些细节您不用理会，只要提供 SQL 与参数就好了。

如果需要批处理，可以实现 org.springframework.jdbc.core.BatchPreparedStatementSetter 接口：

```
package org.springframework.jdbc.core;

import java.sql.PreparedStatement;
import java.sql.SQLException;

public interface BatchPreparedStatementSetter {
   void setValues(PreparedStatement ps,
                  int i) throws SQLException;
   int getBatchSize();
}
```

例如可以在 JdbcTemplateDemo 项目中的 IUserDAO 接口及 UserDAO 类上增加一个 insertUsers() 方法的定义与实现，如以下所示的内容：

```
...
public int[] insertUsers(final List users) {
   String sql = "INSERT INTO user (name,age) VALUES(?,?)";
   BatchPreparedStatementSetter setter =
     new BatchPreparedStatementSetter() {
       public void setValues(
           PreparedStatement ps, int i) throws SQLException {
           User user = (User) users.get(i);
           ps.setString(1, user.getName());
           ps.setInt(2, user.getAge().intValue());
       }
       public int getBatchSize() {
           return users.size();
       }
     };

   return jdbcTemplate.batchUpdate(sql, setter);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}
...
```

如果 JDBC 驱动程序支持批处理，则直接使用它的功能，如果不支持， Spring 则会一个一个自动处理更新以模拟批处理。

5.2.3 JdbcTemplate 查询

使用 JdbcTemplate 进行查询时，可以使用 queryForXXX() 等方法，例如使用 queryForInt()方法传回 user 表格中的数据数目：

```
jdbcTemplate.queryForInt("SELECT COUNT(*) FROM user");
```

也可以使用 queryForObject()传回一个查询后的结果对象，例如传回一个 String 对象：

```
String name = (String) jdbcTemplate.queryForObject(
            "SELECT name FROM USER WHERE id = ?",
             new Object[] {id},
             java.lang.String.class);
```

上面两个例子传回的都是单独一笔数据，如果要传回多笔数据，则可以使用 queryForList() 方法，例如：

```
List rows = jdbcTemplate.queryForList(
         "SELECT * FROM user WHERE id=" + id.intValue());
```

传回的 List 中包括的是 Map 对象，每个 Map 对象代表查询结果中的一笔数据，每笔数据包括多个字段内容，要取得字段中的值，就要使用字段名称作为"键（Key）"，例如：

```
...
Iterator it = rows.iterator();
while(it.hasNext()) {
    Map userMap = (Map) it.next();
    System.out.println(userMap.get("id"));
    System.out.println(userMap.get("name"));
    System.out.println(userMap.get("age"));
    ...
}
...
```

您可以实现 org.springframework.jdbc.core.RowCallbackHandler 接口，在查询到数据之后先作一些处理再传回，例如修改一下 5.2.1 节的 JdbcTemplateDemo 项目中 UserDAO 的 find()方法，如下所示，在 RowCallback-Handler 的 processRow()方法中实现简单的 ORM

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

（Object-Relational Mapping）动作：

```
...
public User find(Integer id) {
    final User user = new User();
    jdbcTemplate.query(
       "SELECT * FROM user WHERE id = ?",
       new Object[] {id},
        new RowCallbackHandler() {
           public void processRow(ResultSet rs)
                                throws SQLException {
              user.setId(new Integer(rs.getInt("id")));
              user.setName(rs.getString("name"));
              user.setAge(new Integer(rs.getInt("age")));
           }
        });

    return user;
}
...
```

如果一次要取回很多查询结果的对象，则可以先实现 org.springframework.jdbc.core.RowMapper 接口，例如：

```
package onlyfun.caterpillar;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class UserRowMapper implements RowMapper {

    public Object mapRow(ResultSet rs,
                         int rowNum) throws SQLException {
        User user = new User();
        user.setId(new Integer(rs.getInt("id")));
        user.setName(rs.getString("name"));
        user.setAge(new Integer(rs.getInt("age")));

        return user;
    }
}
```

在 Spring 2.0 中，ResultReader 接口与其实现类已经被移除，查询方法现在直接接受 RowMapper 实例，例如使用 queryForObject()方法时：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
...
   public User find(Integer id) {
      User user = (User) jdbcTemplate.queryForObject(
            "select * from user where id=?",
            new Object[] {id},
            new UserRowMapper());

      return user;
   }
...
```

传回的结果已使用 UserRowMapper 的定义，将之封装为 User 对象。以下是使用 query() 方法时的一个示范：

```
List users = jdbcTemplate.query("select * from user",
             new UserRowMapper());

for(int i = 0; i < users.size(); i++) {
   User user = (User) users.get(i);
   System.out.println("\tId:\t" + user.getId());
   System.out.println("\tName:\t" + user.getName());
   System.out.println("\tAge:\n" + user.getAge());
}
```

传回的 List 对象中，包括了从数据库中查询出来的结果，并已封装为 User 类的实例。

5.2.4 JdbcTemplate 的 Lob 支持

在 JDBC 中可以使用 Clob（Character large object）与 Blob（Binary large object），以分别针对文字文件与二进制文件进行储存。Spring 中可以通过 JdbcTemplate 来处理 CLOB 与 BLOB，以避免处理特定数据库（例如 Oracle 9i）的 Clob、Blob 差异问题。举个例子来说，假设您的 MySQL 数据库表格如下：

```
CREATE TABLE test (
    id INT AUTO_INCREMENT PRIMARY,
    txt TEXT,
    image BLOB
);
```

假设现在分别读取一个文字文件与二进制文件，并想将之储存至数据库中，则可以使用 JdbcTemplate，例如：

```
final File binaryFile = new File("wish.jpg");
final File txtFile = new File("test.txt");
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
final InputStream is = new FileInputStream(binaryFile);
final Reader reader = new FileReader(txtFile);

JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

final LobHandler lobHandler = new DefaultLobHandler();

jdbcTemplate.execute("INSERT INTO test (txt, image) VALUES(?, ?)",
  new AbstractLobCreatingPreparedStatementCallback(lobHandler) {
    protected void setValues(PreparedStatement pstmt,
      LobCreator lobCreator) throws SQLException,DataAccessException {
        lobCreator.setClobAsCharacterStream(
              pstmt, 1, reader, (int) txtFile.length());
        lobCreator.setBlobAsBinaryStream(
              pstmt, 2, is, (int) binaryFile.length());
    }
 });
reader.close();
is.close();
```

在建立 AbstractLobCreatingPreparedStatementCallback 对象时，要传递一个 LobHandler 实例，对于 MySQL（MS SQL Server 或 Oracle 10g），使用 DefaultLobHandler 即可，对于 Oracle 9i 特定的 LOB 处理，可以使用 OracleLobHandler。在 setValues()方法实现中，使用 LogCreator 来分别设置 Blob 与 Clob 的来源串流，索引 1、2 表示第一与第二个占位字符'?'的位置，并指定读取长度。

如果要从数据库中将数据读取出来，并另存为文件，可以使用以下的程序：

```
final Writer writer = new FileWriter("test_bak.txt");
final OutputStream os = new FileOutputStream(new File(wish_bak.jpg"));

jdbcTemplate.query("SELECT txt,image FROM test WHERE id = ?",
   new Object[] {new Integer(1)},
   new AbstractLobStreamingResultSetExtractor() {
     protected void streamData(ResultSet rs)
           throws SQLException, IOException, DataAccessException {
        FileCopyUtils.copy(
              lobHandler.getClobAsCharacterStream(rs, 1), writer);
        FileCopyUtils.copy(
              lobHandler.getBlobAsBinaryStream(rs, 2), os);
     }
  });
writer.close();
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
os.close();
```

在这里使用 FileCopyUtils 的 copy()方法，将 LogHandler 取得的串流直接转接给文件输出 FileWriter、FileOutputStream 对象。

5.2.5 以对象方式进行操作

就 JdbcTemplate 上的各种方法来看，它封装了 JDBC 处理的细节，让您不用接触底层的数据库技术，然而 JdbcTemplate 的各种方法仍须熟悉如何使用 SQL 语法，如果想让一些程序开发人员不用接触到 SQL 语法或是重用某些 SQL，在 Spring 中，您可以进一步建立可重用的操作对象，建立之后，在设计数据库时，将设计完成的可重用对象给开发人员使用，他们就无须接触到 SQL 的细节了，从他们的观点来看，数据库操作更接近对象导向的操作方式。

Spring 提供了 org.springframework.jdbc.object 包，可以让您以更对象导向的方式来设计数据库操作方面的程序，只要事先继承或直接实例化相对应的类并进行编译，之后就可以重复利用这个实例，执行它的方法来进行数据库相关操作。

org.springframework.jdbc.object.RdbmsOperation 是个抽象类，代表 RDBMS（Relational Database Management System）的查询、更新、预存程序等操作，您可以使用其子类 org.springframework.jdbc.object.SqlUpdate 、 org.springframework.jdbc.object.MappingSqlQuery 等类，它们被设计为线程安全（Thread-safe），所以可以在多线程的环境下重复使用这些类的实例。

SqlFunction 用来执行 SQL Function，例如可以执行 COUNT()，然后回传一个整数表示查询到的数据笔数，下面是一个例子：

```
SqlFunction sf = new SqlFunction(
   dataSource, "SELECT COUNT(*) from user");
sf.compile();
sf.run();
```

RdbmsOperation 的子类在设置好 DataSource、SQL 以及相关参数后，必须先执行 compile()进行编译，之后就可以重复使用这个实例了。在设计时可以继承 SqlFunction 来封装 SQL，例如修改 5.2.1 节中 JdbcTemplateDemo 项目的内容，新增一个类如下所示：

RdbmsDemo　　UserFunction.java

```
package onlyfun.caterpillar;

import javax.sql.DataSource;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.jdbc.object.SqlFunction;

public class UserFunction extends SqlFunction {
    public UserFunction(DataSource dataSource) {
        super(dataSource, "SELECT COUNT(*) from user");
        compile();
    }
}
```

SqlUpdate 类用来表示一个 SQL 的更新操作，您也可以设置相关参数，设计时也可以继承它来进行 SQL 的封装，例如：

RdbmsDemo　　UserUpdate.java

```
package onlyfun.caterpillar;

import java.sql.Types;

import javax.sql.DataSource;

import org.springframework.jdbc.object.SqlUpdate;

public class UserUpdate extends SqlUpdate {
    public UserUpdate(DataSource dataSource) {
        super(dataSource,
              "INSERT INTO user (name,age) VALUES(?,?)");
        int[] types = {Types.VARCHAR, Types.INTEGER};
        setTypes(types);
        compile();
    }
}
```

setTypes()方法用来设置 SQL 中 "?" 占位字符所要插入的数据类型，之后执行 update()方法时，可以仅指定对象数组作为参数来进行查询，每一个数组值将实际取代 "?" 占位字符，例如可以这么使用：

```
...
    SqlUpdate userUpdate = new UserUpdate(dataSource);
    ...
    userUpdate.update(
        new Object[] {user.getName(), user.getAge()});
```

SqlQuery 类表示一个 SQL 查询操作，但通常很少直接使用这个类，因为查询到数据之后，会作一些处理。例如封装 User 类的实例，可以用它的子类 org.spring-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

framework.jdbc.object.MappingSqlMapping 来进行这个动作，例如：

RdbmsDemo UserQuery.java

```
package onlyfun.caterpillar;

import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;

import org.springframework.jdbc.object.MappingSqlQuery;

public class UserQuery extends MappingSqlQuery {
    public UserQuery(DataSource dataSource) {
        super(dataSource, "SELECT * FROM user");
        compile();
    }

    protected Object mapRow(ResultSet rs,
                            int rowNum) throws SQLException {
        User user = new User();

        user.setId(new Integer(rs.getInt("id")));
        user.setName(rs.getString("name"));
        user.setAge(new Integer(rs.getInt("age")));

        return user;
    }
}
```

设计好这个类之后，可以这么使用它：

```
...
SqlQuery userQuery = new UserQuery(dataSource);
...
List users = userQuery.execute();
```

执行完 execute()方法后，实际上传回的 List 中会有 User 类的实例，并且它封装了查询后的每一笔数据。

配合以上的几个实现，可以再改写一下 IUserDAO 接口与 UserDAO 实现，例如：

RdbmsDemo IUserDAO.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.util.List;

public interface IUserDAO {
    public void insert(User user);
    public List allUser();
    public int count();
}
```

假设您是程序开发人员，有其他的开发人员负责数据库存取方面的程序，并提供了以上的 UserFunction、UserUpdate 与 UserQuery 类，那么您就不用关心实际的 SQL 是如何编写的了，您可以重用 UserFunction、UserUpdate 与 UserQuery 类来设计您的 UserDAO 类：

RdbmsDemo　　UserDAO.java

```
package onlyfun.caterpillar;

import java.util.List;

import javax.sql.DataSource;

import org.springframework.jdbc.object.SqlFunction;
import org.springframework.jdbc.object.SqlQuery;
import org.springframework.jdbc.object.SqlUpdate;

public class UserDAO implements IUserDAO {
    private SqlUpdate userUpdate;
    private SqlQuery userQuery;
    private SqlFunction userFunction;

    public void setDataSource(DataSource dataSource) {
        userUpdate = new UserUpdate(dataSource);
        userQuery = new UserQuery(dataSource);
        userFunction = new UserFunction(dataSource);
    }

    public void insert(User user) {
        userUpdate.update(
          new Object[] {user.getName(), user.getAge()});
    }

    public List allUser() {
        return userQuery.execute();
    }

    public int count() {
        return userFunction.run();
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

将 SQL 封装重用之后，从 UserDAO 程序编写的角度来看，完全是用对象操作的方式来进行的，配合程序的修改，可以写个简单的程序来看看运作是否正常：

RdbmsDemo SpringDAODemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;
import java.util.List;

public class SpringDAODemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                   "beans-config.xml");

        User user = new User();
        user.setName("just933");
        user.setAge(new Integer(26));

        IUserDAO userDAO =
            (IUserDAO) context.getBean("userDAO");

        userDAO.insert(user);

        System.out.println("笔数: " + userDAO.count());

        List list = userDAO.allUser();
        for(int i = 0; i < list.size(); i++) {
            User next = (User) list.get(i);
            System.out.println("\n\tId:\t" + next.getId());
            System.out.println("\tName:\t" + next.getName());
            System.out.println("\tAge:\t" + next.getAge());
        }
    }
}
```

良葛格的话匣子<<<

为什么 UserDAO 中的 UserUpdate、UserQuery、UserFunction 等不使用 IoC 依赖注入呢？除了简化范例实现之外，还有一个观念：不要为了 IoC 而 IoC！每个对象不管三七二十一都要来个依赖注入的话，被注入依赖的对象上，将会有一大堆冗长的 Setter 方法。建议有两个以上的对象需要注入相同的依赖时，才考虑使用 Spring IoC 容器来进行依赖注入。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.2.6　DataFieldMaxValueIncrementer

在插入数据至数据库中时，您会需要设置主键，主键的设置最好是与业务键值无关，在 Spring 中，您可以使用 org.springframework.jdbc.support.incrementer.DataFieldMaxValueIncrementer 来为您产生主键值，DataFieldMaxValueIncrementer 有许多针对不同数据库的实例，例如 DB2SequenceMaxValueIncrementer、HsqlMaxValueIncrementer、MySQLMaxValueIncrementer、OracleSequenceMaxValueIncrementer、PostgreSQLSequenceMaxValueIncrementer，您可以操作它的 nextIntValue()、nextLongValue() 或 nextStringValue() 来产生下一个主键值。

5.2.7　Spring 2.0 的 NamedParameterJdbcTemplate

在 Spring 2.0 中新增了 NamedParameterJdbcTemplate 类，这样您在编写 JDBC 的 SQL 陈述时就不必使用占位字符'?'了，而是使用实际的命名参数（Named parameter）来保留 SQL 中会变动的数据部分，例如原本您是这么编写一个查询的：

```
String sql = "SELECT * FROM user WHERE id=?";
List rows = jdbcTemplate.queryForList(sql,
        new Object[] {id.intValue()});
```

现在可以使用 NamedParameterJdbcTemplate 改写为以下的方式：

```
String sql = "SELECT * FROM user WHERE id=:userId";
SqlParameterSource namedParameters =
       new MapSqlParameterSource("userId", id);

NamedParameterJdbcTemplate jdbcTemplate =
             new NamedParameterJdbcTemplate(dataSource);
List rows = jdbcTemplate.queryForList(sql, namedParameters);
```

注意到命名参数":userId"实际值的指定方式，在这里使用的是 SqlParameterSource，在构造时直接指定命名参数与实际值。您也可以使用 Map 对象来指定多个命名参数的实际值，例如：

```
String sql = "INSERT INTO user (name,age) VALUES(:userName, :userAge)";
Map namedParameters = new HashMap();
namedParameters.put("userName", name);
namedParameters.put("userAge", age);

NamedParameterJdbcTemplate jdbcTemplate =
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            new NamedParameterJdbcTemplate(dataSource);
jdbcTemplate.update(sql, namedParameters);
```

您也可以将一个 POJO（Plain Old Java Object）对象的值作为命名参数值的依据，例如：

```
String sql = "INSERT INTO user (name,age) VALUES(:name, :age)";
SqlParameterSource namedParameters =
        new BeanPropertySqlParameterSource(user);
NamedParameterJdbcTemplate jdbcTemplate =
            new NamedParameterJdbcTemplate(dataSource);
jdbcTemplate.update(sql, namedParameters);
```

其中"user"所参考的对象是 JdbcTemplateDemo 项目中的 User 实例。下面具体将 JdbcTemplateDemo 项目的 UserDAO 类改写为使用 NameParameterJdbcTemplate 的方式：

NamedParameterDemo　　UserDAO.java

```
package onlyfun.caterpillar;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import javax.sql.DataSource;
import org.springframework.jdbc.core.namedparam
                    .BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam
                    .MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam
                    .NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;

public class UserDAO implements IUserDAO {
   private NamedParameterJdbcTemplate jdbcTemplate;

   public void setDataSource(DataSource dataSource) {
       jdbcTemplate = new NamedParameterJdbcTemplate(dataSource);
   }

   public void insert(User user) {
      String sql = "INSERT INTO user (name,age) VALUES(:name, :age)";
      SqlParameterSource namedParameters =
          new BeanPropertySqlParameterSource(user);

      jdbcTemplate.update(sql, namedParameters);
   }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public User find(Integer id) {
        String sql = "SELECT * FROM user WHERE id=:userId";
        SqlParameterSource namedParameters =
                new MapSqlParameterSource("userId", id);
        List rows = jdbcTemplate.queryForList(sql, namedParameters);

        Iterator it = rows.iterator();
        if(it.hasNext()) {
            Map userMap = (Map) it.next();

            Integer i = new Integer(userMap.get("id").toString());
            String name = userMap.get("name").toString();
            Integer age =
                 new Integer(userMap.get("age").toString());

            User user = new User();
            user.setId(i);
            user.setName(name);
            user.setAge(age);

            return user;
        }

        return null;
    }
}
```

5.2.8　Spring 2.0 的 SimpleJdbcTemplate

如果您使用的 JDK 是 5.0 以上的版本，则可以利用 Spring 2.0 中 SimpleJdbcTemplate 所提供的泛型（Generic）功能，例如原来您可能是这么查询数据的：

```
public User find(Integer id) {
   String sql = "SELECT * FROM user WHERE id=?";

   RowMapper mapper = new RowMapper() {
       public Object mapRow(ResultSet rs, int rowNum)
                                    throws SQLException {
           User user = new User();
           user.setId(new Integer(rs.getInt("id")));
           user.setName(rs.getString("name"));
           user.setAge(new Integer(rs.getInt("age")));
           return user;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
    };

    JdbcTemplate simpleJdbcTemplate =
         new JdbcTemplate(this.getDataSource());

    return (User) jdbcTemplate.queryForObject(sql, mapper, id);
}
```

若改用 SimpleJdbcTemplate，则可以如下编写：

```
public User find(Integer id) {
   String sql = "SELECT * FROM user WHERE id=?";

   ParameterizedRowMapper<User> mapper =
    new ParameterizedRowMapper<User>() {
       public User mapRow(ResultSet rs, int rowNum)
                                    throws SQLException {
          User user = new User();
          user.setId(new Integer(rs.getInt("id")));
          user.setName(rs.getString("name"));
          user.setAge(new Integer(rs.getInt("age")));
          return user;
       }
    };

    SimpleJdbcTemplate simpleJdbcTemplate =
       new SimpleJdbcTemplate(this.getDataSource());

    return simpleJdbcTemplate.queryForObject(sql, mapper, id);
}
```

可以注意到，当中使用了泛型功能，可以直接在 mapRow()方法之后传回 User 实例，而 SimpleJdbcTemplate 的 queryForObject()方法也可以直接传回查询结果的 User 实例。下面将 JdbcTemplateDemo 项目的 UserDAO 改写为使用 SimpleJdbcTemplate 的方式：

SimpleJdbcTemplateDemo　　UserDAO.java

```
package onlyfun.caterpillar;

import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;
import org.springframework.jdbc.core.simple.ParameterizedRowMapper;
import org.springframework.jdbc.core.simple.SimpleJdbcTemplate;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class UserDAO implements IUserDAO {
    private SimpleJdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        jdbcTemplate = new SimpleJdbcTemplate(dataSource);
    }

    public void insert(User user) {
        String sql = "INSERT INTO user (name,age) VALUES(?, ?)";
        String name = user.getName();
        Integer age = user.getAge();

        jdbcTemplate.update(sql, new Object[] {name, age});
     }

    public User find(Integer id) {
        String sql = "SELECT * FROM user WHERE id=?";

        ParameterizedRowMapper<User> mapper =
          new ParameterizedRowMapper<User>() {
            public User mapRow(ResultSet rs, int rowNum)
                                        throws SQLException {
                User user = new User();
                user.setId(new Integer(rs.getInt("id")));
                user.setName(rs.getString("name"));
                user.setAge(new Integer(rs.getInt("age")));
                return user;
            }
        };

        return jdbcTemplate.queryForObject(sql, mapper, id);
    }
}
```

5.3　JDBC 事务管理

Spring 提供编程式的事务管理（Programmatic transaction manage- ment）与声明式的事务管理（Declarative transaction management），为不同的事务实现提供了一致的编程模型，这节以 JDBC 事务为例，介绍 Spring 的事务管理。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.3.1 Spring 对事务的支持

事务是一组原子（Atomic）操作的工作单元，以数据库存取的实例来说，就是一组 SQL 指令，这一组 SQL 指令必须全部执行成功，若因为某个原因未全部执行成功（例如其中一行 SQL 有错误），则先前所有执行过的 SQL 指令都会被撤消。

举个简单的例子，一个客户从 A 银行转账至 B 银行，要作的动作为从 A 银行的账户扣款、在 B 银行的账户加上转账的金额，两个动作必须成功，如果有一个动作失败，则此次转账失败。

事务还必须保持所参与资源的一致性（Consistent），例如在银行账户的例子中，两个账户的转账金额，B 账户取款的金额不能大于 A 账户的存款金额。每个事务彼此之间必须是隔离的（Isolated），例如在 A 账户中可能有两笔事务，同时进行存款与提款的动作，两个事务基本上不需意识到彼此的存在。事务还必须是可持续的（Durable），在某一笔事务之后，这笔事务必须是被记录下来的。

在这里将介绍 JDBC 如何使用事务管理。首先来看看事务的原子性实现，在 JDBC 中，可以操作 Connection 的 `setAutoCommit()` 方法，给定 `false` 参数，在下达一连串的 SQL 语句后，自行执行 Connection 的 `commit()`来送出变更，如果中间发生错误，则执行 `rollback()` 来撤消所有的执行，例如：

```
try {
    .....
    connection.setAutoCommit(false);
    .....
    // 一连串 SQL 操作
    connection.commit();
} catch(SQLException) {
    // 发生错误，撤消所有变更
    connection.rollback();
}
```

在 Spring 中对 JDBC 的事务管理加以封装，Spring 事务管理的抽象关键在于 `org.springframework.transaction.PlatformTransactionManager` 接口的实现：

```
...
public interface PlatformTransactionManager {
    TransactionStatus getTransaction(TransactionDefinition
                definition)  throws TransactionException;
    void commit(TransactionStatus status)
                            throws TransactionException;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    void rollback(TransactionStatus status)
                            throws TransactionException;
}
```

PlatformTransactionManager 接口有许多具体的事务实现类，例如 DataSourceTransactionManager、HibernateTransactionManager、JdoTransactionManager、JtaTransactionManager 等，通过依赖于 PlatformTransactionManager 接口及各种的技术实现，Spring 在事务管理上可以让开发人员使用一致的编程模型，即使所使用的是不同的事务管理技术。

TransactionException 是 Unchecked Exception。事务的失败通常都是致命的错误，Spring 不强迫您一定要处理，而是让您自行选择是否要捕捉异常。

getTransaction() 方法根据一个 TransactionDefinition 对象来回传一个 TransactionStatus 对象，TransactionDefinition 接口的实例定义了事务的隔离程度（Isolation level）、传播行为（Propagation behavior）、超时（Timeout）、只读（Read-only）等，TransactionStatus 代表着一个新的事务发起或已经存在的事务，您可以通过它来控制事务的执行或调查的状态：

```
...
public interface TransactionStatus {
    boolean isNewTransaction();
    void setRollbackOnly();
    boolean isRollbackOnly();
}
```

Spring 提供编程式的事务管理（Programmatic transaction management）与声明式的事务管理（Declarative transaction management）：

- **编程式的事务管理**

 编程式的事务管理可以清楚地控制事务的边界，也就是让您自行实现事务开始时间、撤消操作的时机、结束时间等，可以实现细粒度的事务控制。

- **声明式的事务管理**

 然而多数的情况下，事务并不需要细粒度的控制，而是采用声明式的事务管理，好处是 Spring 事务管理的相关 API 可以不用介入程序之中，从对象的角度来看，它并不知道自己正被纳入事务管理之中，在不需要事务管理的时候，只要在设置文件上修改一下设置，即可移去事务管理服务。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.3.2 JDBC 编程事务管理

Spring 提供两种方式实现编程式的事务管理，一是直接使用 PlatformTransactionManager 实现，二是使用 org.springframework.transaction.support.TransactionTemplate。

先来看看如何使用 PlatformTransactionManager，在这里使用它的实现类 DataSourceTransactionManager，可以改写一下之前 5.2.1 节中的 JdbcTemplateDemo 项目，让它具有事务管理功能，修改一下 UserDAO 类的 insert() 方法来作示范：

ProgrammaticTransactionDemo　　UserDAO.java

```
package onlyfun.caterpillar;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import javax.sql.DataSource;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.
          datasource.DataSourceTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.
          support.DefaultTransactionDefinition;

public class UserDAO implements IUserDAO {
    private DataSourceTransactionManager transactionManager;
    private DefaultTransactionDefinition def;
    private JdbcTemplate jdbcTemplate;

    public void setDataSource(DataSource dataSource) {
        jdbcTemplate = new JdbcTemplate(dataSource);
        transactionManager =
            new DataSourceTransactionManager(dataSource);
        // 建立事务的定义
        def = new DefaultTransactionDefinition();
        def.setPropagationBehavior(
              TransactionDefinition.PROPAGATION_REQUIRED);
    }

    public void insert(User user) {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        String name = user.getName();
        int age = user.getAge().intValue();

        TransactionStatus status =
            transactionManager.getTransaction(def);
        try {
            jdbcTemplate.update("INSERT INTO user (name,age) "
                    + "VALUES('" + name + "'," + age + ")");
            // 下面的 SQL 有错误，用以测试事务
            jdbcTemplate.update("INSER INTO user (name,age) "
                    + "VALUES('" + name + "'," + age + ")");
        }
        catch(DataAccessException e) {
            transactionManager.rollback(status);
            throw e;
        }
        transactionManager.commit(status);
    }

    public User find(Integer id) {
        List rows = jdbcTemplate.queryForList(
          "SELECT * FROM user WHERE id=" + id.intValue());

        Iterator it = rows.iterator();
        if(it.hasNext()) {
            Map userMap = (Map) it.next();
            Integer i = new Integer(
                    userMap.get("id").toString());
            String name = userMap.get("name").toString();
            Integer age = new Integer(
                    userMap.get("age").toString());

            User user = new User();

            user.setId(i);
            user.setName(name);
            user.setAge(age);

            return user;
        }

        return null;
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在 insert()方法中使用了 DataSourceTransactionManager 来进行事务管理，如果发生了异常，则 catch 区块中会进行事务的 Rollback，在 insert() 方法中故意写入错误的 SQL（注意 INSERT 方法少写了一个 T），因此实际上数据并不会被储存至数据库中。

良葛格的话匣子<<<

要使用 MySQL 数据库进行事务处理，必须建立支持事务的表格类型，例如 InnoDB 的表格类型，这里用来建立表格的 SQL 如下所示：

```
CREATE TABLE user (
  id INT(11) NOT NULL auto_increment PRIMARY KEY,
  name VARCHAR(100) NOT NULL default '',
  age INT
) TYPE = InnoDB;
```

另一个实现编程式事务管理的方法是使用 TransactionTemplate，它需要一个 TransactionManager 实例，如下所示：

```
...
TransactionTemplate transactionTemplate =
       new TransactionTemplate(transactionManager);
...
transactionTemplate.execute(new TransactionCallback() {
   public Object doInTransaction(TransactionStatus status) {
        return jdbcTemplate.update("INSERT INTO user (name,age) "
             + "VALUES('" + name + "'," + age + ")");
   }
});
```

如果发生了异常，则会进行 Rollback，否则提交事务，如果没有回传值，则也可以使用 TransactionCallbackWithoutResult：

```
...
transactionTemplate.execute(
       new TransactionCallbackWithoutResult() {
             public void doInTransactionWithoutResult(
                          TransactionStatus status) {
          ...
             }
          });
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

5.3.3　JDBC 声明事务管理

Spring 声明式的事务管理依赖它的 AOP 框架来完成。使用声明事务管理的好处是，事务管理不能侵入您所开发的组件，具体来说，DAO 对象不会意识到正在事务管理之中，事实上也应当如此，因为事务管理是属于系统层面的服务，而不是业务逻辑的一部分，如果想要改变事务管理策略的话，也只需要在定义文件中重新配置。

举个例子来说，可以将 5.2.1 节中的 JdbcTemplateDemo 项目修改一下，在不修改 UserDAO 类的情况下，可以为它加入事务管理的服务，一个简单的方法是使用 TransactionProxyFactoryBean，指定要介入的事务管理对象及其方法，这需要在定义文件中修改，如下所示：

DeclarativeTransactionDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                → datasource.DriverManagerDataSource"
         destroy-method="close">
       <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
       <property name="url"
                value="jdbc:mysql://localhost:3306/demo"/>
       <property name="username" value="caterpillar"/>
       <property name="password" value="123456"/>
    </bean>

    <bean id="transactionManager"
         class="org.springframework.jdbc.
                → datasource.DataSourceTransactionManager">
       <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="userDAO"
         class="onlyfun.caterpillar.UserDAO">
       <property name="dataSource" ref="dataSource"/>
    </bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="userDAOProxy"
          class="org.springframework.transaction.
              → interceptor.TransactionProxyFactoryBean">
        <property name="proxyInterfaces">
            <list>
                <value>onlyfun.caterpillar.IUserDAO</value>
            </list>
        </property>
        <property name="target" ref="userDAO"/>
        <property name="transactionManager"
                  ref="transactionManager"/>
        <property name="transactionAttributes">
            <props>
                <prop key="insert*">PROPAGATION_REQUIRED</prop>
            </props>
        </property>
    </bean>

</beans>
```

TransactionProxyFactoryBean 需要一个 TransactionManager，由于这里使用的是 JDBC，所以使用 DataSourceTransactionManager，TransactionProxyFactoryBean 是个代理对象，"target" 属性指定要代理的对象，事务管理会自动介入指定的方法前后，这里使用 "transactionAttributes" 属性指定，"insert*" 表示指定方法名称以 insert 开头的都要纳入事务管理，您也可以指定方法全名，如果在方法执行过程中发生错误，则所有先前的操作自动撤回，否则正常提交。

在"insert*" 等方法上指定了 "PROPAGATION_REQUIRED"，表示在目前的事务中执行操作，如果事务不存在就建立一个新的，相关的常数意义都可以在 API 文件的 TransactionDefinition 接口中找到。您可以加上多个事务定义，中间使用逗号 "," 区隔，例如可以加上只读，或者是指定某个异常发生时撤回操作：

```
PROPAGATION_REQUIRED,readOnly,-MyCheckedException
```

MyCheckedException 前面加上 "-" 时，表示发生指定异常时撤消操作，如果前面加上 "+"，表示发生异常时立即提交。

由于"userDAO"被"userDAOProxy"代理了，所以要做的是取得"userDAOProxy"，而不是"userDAO"，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

DeclarativeTransactionDemo　　SpringDAODemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringDAODemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "beans-config.xml");

        User user = new User();

        user.setName("caterpillar");
        user.setAge(new Integer(30));

        IUserDAO userDAO =
            (IUserDAO) context.getBean("userDAOProxy");

        userDAO.insert(user);

        user = userDAO.find(new Integer(1));

        System.out.println("name: " + user.getName());
    }
}
```

您也可以设置不同的 TransactionInterceptor 来得到更多的管理细节，例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                 → datasource.DriverManagerDataSource"
         destroy-method="close">
        <property name="driverClassName"
                 value="com.mysql.jdbc.Driver"/>
        <property name="url"
                 value="jdbc:mysql://localhost:3306/demo"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <property name="username" value="caterpillar"/>
        <property name="password" value="123456"/>
    </bean>

    <bean id="transactionManager"
         class="org.springframework.jdbc.
                → datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="userDAO"
         class="onlyfun.caterpillar.UserDAO">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="transactionInterceptor"
         class="org.springframework.transaction.
                → interceptor.TransactionInterceptor">
        <property name="transactionManager" ref="transactionManager"/>
        <property name="transactionAttributeSource"
                value="onlyfun.caterpillar.UserDAO.insert*=
                        → PROPAGATION_REQUIRED "/>
    </bean>

    <bean id="userDAOProxy"
         class="org.springframework.aop.
                → framework.ProxyFactoryBean">
        <property name="proxyInterfaces">
            <list>
                <value>onlyfun.caterpillar.IUserDAO</value>
            </list>
        </property>
        <property name="target" ref="userDAO"/>
        <property name="interceptorNames">
            <list>
                <value>transactionInterceptor</value>
            </list>
        </property>
    </bean>
</beans>
```

即使后来不再需要事务管理，也可以直接在 Bean 定义文件中修改配置，而不用修改程序重新进行编译等动作。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

良葛格的话匣子<<<

声明事务管理是利用 Spring AOP 来达成的，所以执行以上的程序时，请记得您的 Classpath 设置中必须包括 spring-aop.jar。

5.3.4　事务的属性介绍

Spring 使用 AOP 来完成声明式的事务管理，因而声明式事务是以方法为边界的，Spring 的事务属性（Transaction attribute）自然就在于描述事务应用至方法上的策略，在 Spring 中事务属性分作以下的几个参数：

- **传播行为（Propagation behavior）**

 传播行为定义了事务应用于方法上之边界（Boundaries），它告知何时该开始一个新的事务，或何时事务该被暂停，或方法是否要在事务中进行。

 Spring 定义了几个传播行为，可以在 `TransactionDefinition` 的 API 文件说明上找到相对应的常数与说明，以下列出几个：

表 5.1　事务传播行为说明

传播行为	说明
PROPAGATION_MANDATORY	方法必须在一个现存的事务中进行，否则丢出异常
PROPAGATION_NESTED	在一个嵌入的事务中进行，如果不是，则同 PROPAGATION_REQUIRED
PROPAGATION_NEVER	指出不应在事务中进行，如果有就丢出异常
PROPAGATION_NOT_SUPPORTED	指出不应在事务中进行，如果有就暂停现存的事务
PROPAGATION_REQUIRED	支持现在的事务，如果没有就建立一个新的事务
PROPAGATION_REQUIRES_NEW	建立一个新的事务，如果现存一个事务就暂停它
PROPAGATION_SUPPORTS	支持现在的事务，如果没有就以非事务的方式执行

举个例子来说，如果传播行为被声明为 `PROPAGATION_REQUIRED`，则事务的边界在开始第一个事务的方法呼叫及结束时，如果先前没有事务被开始，则事务边界即为目前方法的执行前后。又如果传播行为被声明为 `PROPAGATION_REQUIRES_NEW`，则事务的边界即为该方法执行的前后。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- 隔离层级（Isolation level）

 在一个应用程序中，可能有多个事务同时在进行，这些事务应当彼此之间互相不知道另一个事务的存在，好比现在整个应用程序就只有一个事务存在，由于事务彼此之间独立，若读取的是同一个数据的话，就容易发生问题，例如：

 - Dirty read

 某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个 Roll back 了操作，则后一个事务所读取的数据就会是不正确的。

 - Non-repeatable read

 在一个事务的两次查询之中数据不一致，这可能是因为两次查询过程中间插入了一个事务更新的原有的数据。

 - Phantom read

 在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列（Row）数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

为了避免以上问题的方法之一，需要在某个事务进行过程中锁定正在更新或查询的数据字段，直到目前的事务完成，然而完全锁定字段时，若另一个事务来查询同一份数据就必须等待，直到前一个事务完成并解除锁定为止，因而会造成应用程序在查询或更新数据时效率上的问题，而事实上根据需求的不同，并不用在事务进行时完全地锁定数据，隔离层级可以让您根据实际的需求，对数据的锁定进行设置。

Spring 提供了几种隔离层级设置，同类型的设置可以在 `TransactionDefinition` 的 API 文件说明上找到相对应的常数与说明，以下列出几个：

表 5.2 事务隔离层级说明

隔离层级	说明
ISOLATION_DEFAULT	使用底层数据库预设的隔离层级
ISOLATION_READ_COMMITTED	允许事务读取其他并行的事务已经送出（Commit）的数据字段，可以防止 Dirty read 问题
ISOLATION_READ_UNCOMMITTED	允许事务读取其他并行的事务还没送出的数据，会发生 Dirty、Nonrepeatable、Phantom read 等问题

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

续表

隔离层级	说明
ISOLATION_REPEATABLE_READ	要求多次读取的数据必须相同，除非事务本身更新数据，可防止 Dirty、Nonrepeatable read 问题
ISOLATION_SERIALIZABLE	完整的隔离层级，可防止 Dirty、Nonrepeatable、Phantom read 等问题，会锁定对应的数据表格，因而有效率问题

- 只读提示（Read-only hints）

 如果事务只进行读取的动作，则可以利用底层数据库在只读操作时发生的一些最佳化动作，由于这个动作利用到数据库在只读的事务操作最佳化，因而必须在事务中才有效，也就是说要搭配传播行为 PROPAGATION_REQUIRED、PROPAGATION_REQUIRES_NEW、PROPAGATION_NESTED 来设置。

- 事务超时期间（The transaction timeout period）

 有的事务操作可能延续很长一段的时间，事务本身可能关联到数据表格的锁定，因而长时间的事务操作会有效率上的问题，对于过长的事务操作，您要考虑 Roll back 事务并要求重新操作，而不是无限时的等待事务完成。

您可以设置事务超时期间，计时是从事务开始时，所以这个设置必须搭配传播行为 PROPAGATION_REQUIRED、PROPAGATION_REQUIRES_NEW、PROPAGATION_NESTED 来设置。

5.3.5　TransactionAttributeSource、TransactionAttribute

在 TransactionProxyFactoryBean 上有 setTransactionAttributeSource() 与 setTransaction Attributes()方法，它们是用来设置事务属性的策略实例。

org.springframework.transaction.interceptor.TransactionAttributeSource 接口上有一个 getTransactionAttribute() 方法，您可以根据传递给它的 Method 实例与 Class 实例，决定该回传一个什么内容的 org.springframework.transaction.interceptor.TransactionAttribute 实例，一个最简单的 TransactionAttributeSource 实现是 org.springframework.transaction.interceptor.MatchAlwaysTransactionAttributeSource，对于每一个方法执行都会应用事务，它回传的 TransactionAttribute 实例的默认传播行为是 PROPAGATION_REQUIRED，隔离层级为 ISOLATION_DEFAULE。

一个应用的例子如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
...
<bean id="transactionAttributeSource"
     class="org.springframework.transaction.interceptor.
          → MatchAlwaysTransactionAttributeSource"/>

<bean id="userDAOProxy"
     class="org.springframework.transaction.
          → interceptor.TransactionProxyFactoryBean">
   <property name="proxyInterfaces">
      <list>
         <value>onlyfun.caterpillar.IUserDAO</value>
      </list>
   </property>
   <property name="target" ref="userDAO"/>
   <property name="transactionManager" ref="transactionManager"/>
   <property name="transactionAttributeSource"
           ref="transactionAttributeSource"/>
</bean>
...
```

您可以使用 org.springframework.transaction.interceptor.DefaultTransaction-Attribute，并设置自己的事务策略，之后设置给 TransactionAttributeSource，例如：

```
...
<bean id="myTransactionAttribute"
     class="org.springframework.transaction.
        → interceptor.DefaultTransactionAttribute">
   <property name="propagationBehaviorName"
           value="PROPAGATION_REQUIRES_NEW"/>
   <property name="isolationLevelName"
           value="ISOLATION_REPEATABLE_READ"/>
</bean>

<bean id="transactionAttributeSource"
     class="org.springframework.transaction.
     → interceptor.MatchAlwaysTransactionAttributeSource">
   <property name="transactionAttribute"
           ref="myTransactionAttribute"/>
</bean>

<bean id="userDAOProxy"
     class="org.springframework.transaction.
          → interceptor.TransactionProxyFactoryBean">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <property name="proxyInterfaces">
        <list>
            <value>onlyfun.caterpillar.IUserDAO</value>
        </list>
    </property>
    <property name="target" ref="userDAO"/>
    <property name="transactionManager" ref="transactionManager"/>
    <property name="transactionAttributeSource"
              ref="transactionAttributeSource"/>
</bean>
...
```

可以使用 org.springframework.transaction.interceptor.NameMatchTransaction-AttributeSource 来指定某些方法要应用事务，以及要应用的事务策略，例如：

```
...
<bean id="transactionAttributeSource"
      class="org.springframework.transaction.
        → interceptor.NameMatchTransactionAttributeSource">
    <property name="properties">
        <props>
            <prop key="insert*">PROPAGATION_REQUIRES_NEW</prop>
        </props>
    </property>
</bean>

<bean id="userDAOProxy"
      class="org.springframework.transaction.
          → interceptor.TransactionProxyFactoryBean">
    <property name="proxyInterfaces">
        <list>
            <value>onlyfun.caterpillar.IUserDAO</value>
        </list>
    </property>
    <property name="target" ref="userDAO"/>
    <property name="transactionManager" ref="transactionManager"/>
    <property name="transactionAttributeSource"
              ref="transactionAttributeSource"/>
</bean>
...
```

在 NameMatchTransactionAttributeSource 的 "properties"属性上，可以指定方法名称与事务策略，方法名称的指定可以指定全名，也可以使用 Wildcard 来指定，例如上面的指定中，只要方法名称以 insert 为开头的都会应用相对应的事务策略。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在指定事务策略时，指定的格式如下：

传播行为，隔离层级，只读，+异常， -异常

除了传播行为一定要设置之外，其他都可选择性的设置，中间以逗号区隔，例如：

```
PROPAGATION_REQUIRED,readOnly,-MyCheckedException
```

MyCheckedException 前面加上 "-" 时，表示发生指定异常时撤消操作，如果前面加上 "+"，表示发生异常时立即提交。

在比较简单的设置中，可以仅设置 TransactionProxyFactoryBean，并在它的 "transactionAttributes" 属性上直接设置要应用事务的方法及事务策略，例如：

```
...
<bean id="userDAOProxy"
     class="org.springframework.transaction.
        → interceptor.TransactionProxyFactoryBean">
   <property name="proxyInterfaces">
      <list>
         <value>onlyfun.caterpillar.IUserDAO</value>
      </list>
   </property>
   <property name="target" ref="userDAO"/>
   <property name="transactionManager" ref="transactionManager"/>
   <property name="transactionAttributes">
      <props>
         <prop key="insert*">PROPAGATION_REQUIRED</prop>
      </props>
   </property>
</bean>
...
```

甚至也可以直接指定 TransactionInterceptor，以获得更多的控制，例如：

```
...
<bean id="transactionInterceptor"
     class="org.springframework.transaction.
            → interceptor.TransactionInterceptor">
   <property name="transactionManager">
           ref="transactionManager"/>
   <property name="transactionAttributeSource"
             value="onlyfun.caterpillar.UserDAO.insert*=
                  → PROPAGATION_REQUIRED"/>
</bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<bean id="userDAOProxy"
     class="org.springframework.aop.
        → framework.ProxyFactoryBean">
   <property name="proxyInterfaces">
      <list>
         <value>onlyfun.caterpillar.IUserDAO</value>
      </list>
   </property>
   <property name="target" ref="userDAO"/>
   <property name="interceptorNames" value="transactionInterceptor"/>
</bean>
...
```

选择哪一种设置方式是需求的问题，您可以尝试在 DeclarativeTransactionDemo 项目的 Bean 定义文件上设置以上所介绍的方式，基于篇幅的限制，以上仅列出部分的设置内容。

5.3.6　Spring 2.0 声明式事务管理：基于 XML Schmea

在 Spring 2.0 中要设置声明式事务管理，可以依赖于 Spring 2.0 的<aop>与<tx>标签，因而要记得加入相关的名称空间声明：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">
    ...
</beans>
```

事务是系统层面的服务，也就是一个 Aspect，其实具体来说就是一个 Advice，您可以使用<tx:advice>标签来提供这个 Advice，它需要设置一个 TransactionManager，并在当中使用<tx:attributes>来设置事务相关属性。

可以将先前的 DeclarativeTransactionDemo 项目改写，修改其 *beans-config.xml* 为使用<aop>与<tx>标签的方式：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

DeclarativeTransactionDemo2 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                → datasource.DriverManagerDataSource"
         destroy-method="close">
        <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
        <property name="url"
                value="jdbc:mysql://localhost:3306/demo"/>
        <property name="username" value="caterpillar"/>
        <property name="password" value="123456"/>
    </bean>

    <bean id="transactionManager"
         class="org.springframework.jdbc.
                → datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="userDAO"
         class="onlyfun.caterpillar.UserDAO">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <tx:advice id="txAdvice"
              transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="insert*" propagation="REQUIRED"/>
            <tx:method name="find*" read-only="true"/>
        </tx:attributes>
    </tx:advice>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <aop:config>
        <aop:pointcut id="userDAOPointcut"
      expression="execution(* onlyfun.caterpillar.IUserDAO.*(..))"/>
        <aop:advisor advice-ref="txAdvice"
                     pointcut-ref="userDAOPointcut"/>
    </aop:config>

</beans>
```

注意到<tx:method>中的属性设置，对于传播行为、隔离层级、只读、超时、异常时撤回或提交，都有对应的"propagation"、"isolation"、"timeout"、"read-only"、"rollback-for"、"no-rollback-for"属性可以设置，若不设置，"propagation"属性默认是"REQUIRE","isolation"属性默认是"DEFAULT"、"timeout"属性默认是"-1"(单位是秒)、"read-only"属性默认是"false"。

与先前介绍 Spring 2.0 基于 XML Schema 的 AOP 设置相同，由于不再于设置文件中设置代理对象，所以直接取得"userDAO"实例进行操作即可。

5.3.7　Spring 2.0 声明式事务管理：基于 Annotation

声明式事务管理在 Spring 2.0 中，也支持使用 Annotation 的标示方式，方法是使用 @Transactional 来标示，例如可以将 DeclarativeTransactionDemo 项目的 UserDAO 改写，在上头直接标示@Transactional，并设置相关属性：

DeclarativeTransactionDemo3　　UserDAO.java

```
package onlyfun.caterpillar;

import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

public class UserDAO implements IUserDAO {
    private JdbcTemplate jdbcTemplate;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void setDataSource(DataSource dataSource) {
        jdbcTemplate = new JdbcTemplate(dataSource);
    }

    @Transactional(propagation = Propagation.REQUIRED)
    public void insert(User user) {
       String name = user.getName();
       int age = user.getAge().intValue();

       jdbcTemplate.update("INSERT INTO user (name,age) "
              + "VALUES('" + name + "'," + age + ")");
    }

    @Transactional(readOnly=true)
    public User find(Integer id) {
        List rows = jdbcTemplate.queryForList(
          "SELECT * FROM user WHERE id=" + id.intValue());

        Iterator it = rows.iterator();
        if(it.hasNext()) {
            Map userMap = (Map) it.next();

            Integer i = new Integer(userMap.get("id").toString());
            String name = userMap.get("name").toString();
            Integer age =
                 new Integer(userMap.get("age").toString());

            User user = new User();
            user.setId(i);
            user.setName(name);
            user.setAge(age);

            return user;
        }

        return null;
    }
}
```

在使用@Transactional 时，相关的属性设置为"propagation"、"isolation"、"readOnly"、"timeout"、"rollbackFor"、"noRollbackFor"等，而在 *beans-config.xml* 中，则要使用<tx:annotation-driven>标签，并指定 TransactionManager，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

DeclarativeTransactionDemo3　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                → datasource.DriverManagerDataSource"
         destroy-method="close">
        <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
        <property name="url"
                value="jdbc:mysql://localhost:3306/demo"/>
        <property name="username" value="caterpillar"/>
        <property name="password" value="123456"/>
    </bean>

    <bean id="transactionManager"
         class="org.springframework.jdbc.
                → datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="userDAO"
         class="onlyfun.caterpillar.UserDAO">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
```

同样的，由于不再于设置文件中设置代理对象，所以直接取得"userDAO"实例进行操作即可。

5.4　接下来的主题

这节所介绍的是 Spring 在持久层、JDBC、事务等议题的封装与简化，Spring 提供了持久层一致性的设计模型，即使在其他持久层框架的整合上，Spring 结合这些框架来使用，在操

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

作上都很类似。在下一个章节中，将简介一下 Hibernate 的使用，以及看看如何在 Spring 中整合 Hibernate。



图 5.3 Spring 在持久层提供一致性的使用模型

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第

Hibernate 与 Spring

章

Hibernate 是个对象关系映射（Object/Relation Mapping, ORM）的解决方案，提供 Java 对象模型与关系型数据库关联模型的自动映射，也提供有持久层所需的缓存、锁定、事务管理等功能。

您可以在 Spring 中整合 Hibernate，Spring 为 Hibernate 进行了封装，让您在 Hibernate 的设置与使用上，都可以依循与 Spring 一致的使用模型，在使用上可以更为简化，并可以结合 Spring 的功能，如声明式的事务管理等功能。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

6.1　Hibernate 入门

这节是针对未接触过 Hibernate 的读者所安排的，目的在快速了解了如何编写第一个 Hibernate 程序后，以便于在接下来的 6.2 节中，实际练习 Spring 与 Hibernate 的整合范例，如果您已经了解 Hibernate 的编写方式，可以直接略过这节。

6.1.1　简介 Hibernate

在第 5 章中 5.1.2 节的 DataSourceDemo 项目中，曾经设计过一个 UserDAO 类，其中在 find() 方法中，您是这么设计的：

```
...
   public User find(Integer id) {
      Connection conn = null;
      PreparedStatement stmt = null;

       try {
          conn = dataSource.getConnection();
          stmt = conn.prepareStatement(
                    "SELECT * FROM user WHERE id=?");
          stmt.setInt(1, id.intValue());

          ResultSet result = stmt.executeQuery();
          if(result.next()) {
             Integer i = new Integer(result.getInt(1));
             String name = result.getString(2);
             Integer age = new Integer(result.getInt(3));

             User user = new User();
             user.setId(i);
             user.setName(name);
             user.setAge(age);

             return user;
          }
       } catch (SQLException e) {
          e.printStackTrace();
       }
       finally {
          if(stmt != null) {
             try {
                stmt.close();
             }
             catch(SQLException e) {
                e.printStackTrace();
             }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            }
            if(conn != null) {
                try {
                    conn.close();
                }
                catch(SQLException e) {
                    e.printStackTrace();
                }
            }
        }

        return null;
    }
...
```

这是个直接使用 JDBC 进行查询的程序，其中有几个议题是值得探讨的：

- **程序中的 SQL 语句**

 程序中直接编写 SQL 语句，SQL 是数据库层面的语法，Java 则是个支持对象导向模型的程序语言，它们是两种不同的语言模型，另一方面，在程序中穿插 SQL 一直有难以维护的问题。

- **数据对象的封装**

 从数据库查询回来的数据，必须自己编写一些程序，才能将数据封装成 `User` 类的一个实例，当您取得一个 `User` 对象，并想将之储存至数据库中对应的表格时，也要一个一个取出 `User` 对象的数据成员，安插至 SQL 语句之中，再使用 JDBC 的 API 执行储存动作，DataSourceDemo 项目中 `UserDAO` 类的 `insert()`方法就进行了这些动作。

- **异常的处理**

 为了处理执行 JDBC 相关 API 时的异常，整个程序中充满了 try...catch 异常处理逻辑，增加了程序编写时所花费的心力。

- **Connection、Statement 的管理**

 您必须要注意到 Connection 在被使用完毕之后是否关闭，或是有设置连接池（Connection pool）的话，是否将 Connection 放回连接池，以及您也许会考虑到 Statement 要不要作缓存、要不要关闭等，这些问题都要自行解决。

在第 5 章中介绍了 Spring 于 JDBC 的封装与简化，当时介绍的内容可以解决上面所提到的一些议题，但是也有未解决的部分，例如在第 5 章的一些项目范例中仍可以看到，在储存

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

User 对象时，要自行取得 User 对象上的数据成员，安插至 SQL 中以进行数据的储存，从数据库中查询到数据时，也必须自己编写程序来将查询到的数据封装为 User 对象的实例。

事实上由于对象模型与关联模型的不同所引发的问题，还不只这些，例如在 Java 中还有继承、关联、集合等的关系，它们在关系型数据库中要如何作对应，都还有更多的问题有待解决。

Hibernate 是“对象/关系映射”（Object/Relational Mapping）的解决方案，简写为 ORM。简单地说，Hibernate 的作用之一，就是将 Java 中的对象与对象的关系，映射至关系型数据库中的表格与表格之间的关系上，Hibernate 提供了这个过程中自动映射转换的方案。

2001 年末，Hibernate 第一个版本发布，2003 年 6 月 8 日 Hibernate 2 发布，并于年末获得 Jolt 2004 大奖，后被 JBoss（*http://www.jboss.com/*）收纳而成为子项目之一，2005 年 3 月 Hibernate 3 正式发布，这节中所要进行的简介即使用 Hibernate 3，看看如何使用 Hibernate 来改写第 5 章中所设计过的 UserDAO 类，以初步了解 Hibernate 的功能。

6.1.2　下载、设置 Hibernate

编写此书的时候，Hibernate 最新的版本是 3.2 GA（General Availabity），请连接至 Hibernate 的官方网站进行文件的下载，网址是在 *http://hibernate.org/*，如图 6.1 所示。

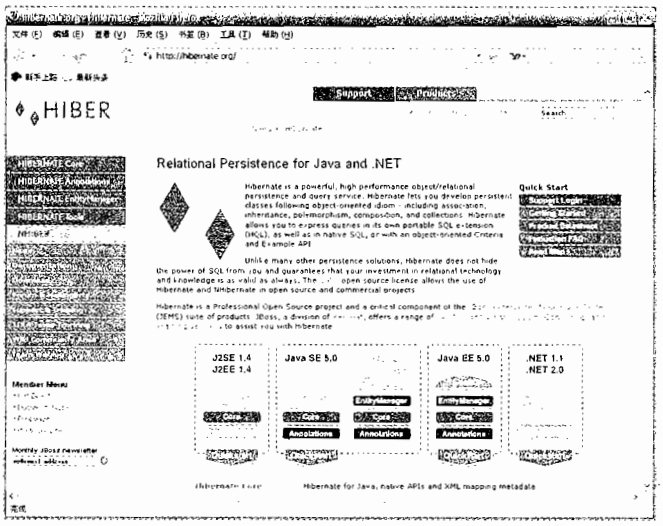


图 6.1　Hibernate 官方网址

点击左边的“Download”链接，可以进入至下载页面，下载页面中会有个“Hibernate Core”

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的“Download”链接连接至 Sourceforge（*http://sourceforge.net/project/showfiles.php?group_id=40712&package_id=127784*），如图 6.2 所示。

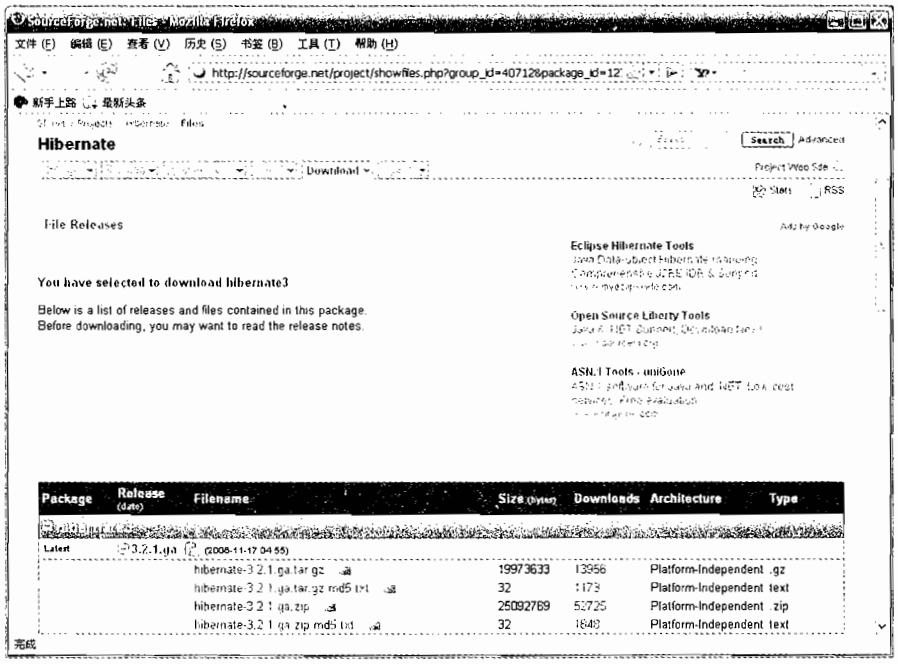


图 6.2 从 Sourceforge 下载 Hibernate 相关文件

在 Sourceforge 上提供有几种不同压缩格式的下载文件，您可以下载 *hibernate-3.2.0.ga.zip*，选择文件的下载链接之后，会需要选择镜像（mirror）网站，可以选择一个最接近下载所在区域的镜像网站，以节省下载的时间。

下载 zip 文件并解压缩，在解开的目录下就有 *hibernate3.jar*，这是 Hibernate 3 所需要的 API 的封装，可以在 *doc* 目录中找到 API 说明或相关参考文件，在 *lib* 目录中的则是可能会用到的相依赖文件。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

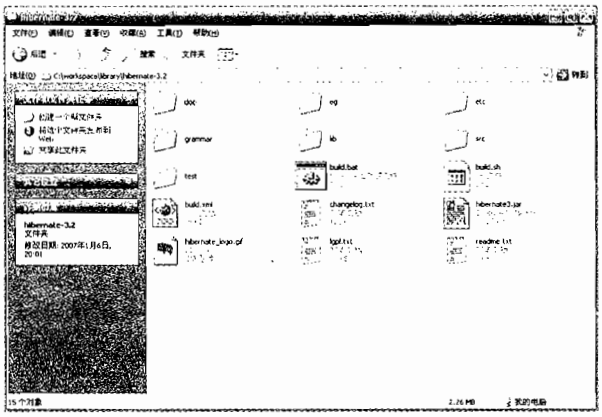


图 6.3　hibernate-3.2 的内容

良葛格的话匣子<<<

为了统一操作的环境，假设您的 Hibernate 相关文件都是放在 *C:\workspace\library\hibernate-3.2* 中的，请参考图 6.3 的内容。

对于即将编写的第一个 Hibernate 程序，所需要的 *.jar* 文件除了 *hibernate3.jar* 之外，您还需要：

- antlr-2.7.6.jar
- asm.jar
- cglib-2.1.3.jar
- commons-collections-2.1.1.jar
- commons-logging-1.0.4.jar
- dom4j-1.6.1.jar
- ehcache-1.2.jar
- jta.jar
- log4j-1.2.11.jar

Hibernate 在底层仍使用 JDBC，所以也必须要有 JDBC 驱动程序的 *jar* 文件，您可以参考第 2 章中 2.1.2 节的内容，在 Eclipse 中将 Hibernate 相关 *.jar* 设置至项目之中。

6.1.3　第一个 Hibernate 程序

直接来编写个项目，示范一下 Hibernate 的功能与使用方式，在这个示范中暂时不使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Spring 的功能，稍后会再来改写它，以了解 Spring 与 Hibernate 整合之后的方便性。

同样的，先设计一个 User 类，这个 User 类将对应至 demo 数据库中的一个 user 表格，它只是一个简单的 POJO（Plain Old Java Object）：

HibernateDemo　　User.java

```
package onlyfun.caterpillar;

public class User {
    private Integer id;
    private String name;
    private Integer age;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

User 类必须有一个 id 属性，用来作为对象在程序中唯一性的识别，这个 id 值可以由 Hibernate 产生，通常对应至数据库表格中的主键值。

良葛格的话匣子<<<

在这里使用的是 MySQL 数据库，表格建立时是使用以下的 SQL:

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
CREATE TABLE user (
    id INT(11) NOT NULL auto_increment PRIMARY KEY,
    name VARCHAR(100) NOT NULL default '',
    age INT
) TYPE = InnoDB;
```

User 类与 user 表格之间要建立对映关系，实际上是靠一个映射文件来完成的，映射文件可以使用 XML 来编写，通常命名为*.hbm.xml*，例如设计一个 *User.hbm.xml* 映射文件：

HibernateDemo　　User.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="onlyfun.caterpillar.User" table="user">

        <id name="id" column="id">
            <generator class="native"/>
        </id>

        <property name="name" column="name"/>

        <property name="age" column="age"/>

    </class>
</hibernate-mapping>
```

在映射文件中，<class> 标签用来定义类与表格的对映，"name" 属性用来设置类名称，"table" 属性用来设置表格名称；<id> 标签中 "name" 属性用来设置对象的唯一识别，通常对应至数据表中的主键字段，也就是 "column" 属性所设置的对应字段；<generator> 的 "class" 用来设置主键的产生方式，设置为 "native" 表示依数据库自己的主键生成方式来生成主键值，例如 user 表格在建立时，主键设置为 auto_increment，所以每加入一笔数据时都会自动递增主键值。

接下来的 <property> 标签则设置 User 对象的每一个属性（由 "name" 属性设置）分别对应至数据库中 user 表格的哪一个字段（由 "column" 属性设置）。

接着以同样的方式，设计一个 IUserDAO 接口，让应用程序在存取数据库时，依赖于抽象的接口而非实际的类实现，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

HibernateDemo IUserDAO.java

```
package onlyfun.caterpillar;

public interface IUserDAO {
    public void insert(User user);
    public User find(Integer id);
}
```

然后由 UserDAO 类来实现 IUserDAO 接口，这次实现时使用 Hibernate 的 API，需要注入一个 org.hibernate.SessionFactory 对象：

HibernateDemo UserDAO.java

```
package onlyfun.caterpillar;

import org.hibernate.Hibernate;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

public class UserDAO implements IUserDAO {
    private SessionFactory sessionFactory;

    public UserDAO() {
    }

    public UserDAO(SessionFactory sessionFactory) {
        this.setSessionFactory(sessionFactory);
    }

    public void setSessionFactory(
                    SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    public void insert(User user) {
        // 取得 Session
        Session session = sessionFactory.openSession();
        // 开启事务
        Transaction tx= session.beginTransaction();
        // 直接储存对象
        session.save(user);
        // 送出事务
        tx.commit();
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        session.close();
    }

    public User find(Integer id) {
        Session session = sessionFactory.openSession();

        User user = (User) session.get(User.class, id);
        Hibernate.initialize(user);

        session.close();

        return user;
    }
}
```

在 Hibernate 中，SessionFactory 用来管理 org.hibernate.Session，Session 的作用类似于 JDBC 中的 Connection，在 Hibernate 中每次进行数据存取之前都必须开启 Session，在进行数据变更或储存时，则必须开启事务，在储存数据的时候，可以使用 Session 的 save() 方法直接将对象储存，在 org.hibernate.Transaction 执行了 commit() 方法之后，Hibernate 会根据映射文件，将对象中的各个属性数据转换，并储存至 user 表格中对应的字段。

在不使用数据库联机的时候，要记得使用 close() 方法关闭 Session，关闭 Session 并不等于关闭 Connection，如果在 Hibernate 中有使用连接池（Connection pool），则在 Session 的 close() 方法执行完毕后，会将 Connection 归还至连接池之中，而不是直接关闭对数据库的连结。

而在 UserDAO 的 find() 方法实现中，可以直接使用 Session 的 load() 方法，指定数据的 id 值与封装数据的类型，接着查询回来的对象中是否包括了相对应的数据，并将之转换回 User 类型，然后就可以操作 User 对象上的各种方法来取得数据。

Hibernate 3 及之后的版本默认是启用延迟加载（Lazy initialization）的功能，也就是在操作对象的 Getter 真正取得数据前，不会向数据表格的对应字段进行查询，而此时 Session 必须是处于开启的状态，否则会丢出 LazyInitializationException。在 find()方法结束前，已经关闭了 Session，为了操作传回的对象以顺利取得对应的字段数据，必须先对 User 实例的每个属性作初始化加载的动作，这可以通过 Hibernate.initialize()方法来完成。

在使用 Hibernate 时，还必须定义一个设置文件放置在 *Classpath* 之下，告知数据库

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

名称、用户名称、密码、映射文件位置等，可以使用 XML 文件来定义，文件名为 *hibernate.cfg.xml*，如下所示：

HibernateDemo hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/
            → hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="show_sql">
            true
        </property>

        <property name="dialect">
            org.hibernate.dialect.MySQLDialect
        </property>

        <property name="connection.driver_class">
            com.mysql.jdbc.Driver
        </property>

        <property name="connection.url">
            jdbc:mysql://localhost/demo
        </property>

        <property name="connection.username">
            caterpillar
        </property>

        <property name="connection.password">
            123456
        </property>

        <!-- 以下设置对象与数据库表格映射文件 -->
        <mapping resource="onlyfun/caterpillar/User.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

在 <session-factory> 的设置中，"show_sql" 被设置为"true"，表示在执行数据库存取动作时显示 Hibernate 所生成、使用的 SQL 语法；Hibernate 可以使用许多不同的数据库，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

不同的数据库在 SQL 或操作上会有所差别，可以设置 "dialect" 来表示将使用哪一种数据库适用的语句操作，由于这里所使用的是 MySQL 数据库，所以设置为 org.hibernate.dialect. MySQLDialect；接下来的 "connection.driver_class"、"connection.url"、"connection. username"、"connection.password" 分别用以设置 JDBC 驱动程序类、URL、用户名称与密码。

在 <mapping> 中的 "resource" 设置的是对象与数据库表格的映射文件位置，在上面的设置为 onlyfun/caterpillar/User.hbm.xml，表示文件在 *Classpath* 路径下 *onlyfun* 文件夹的 *caterpillar* 文件夹下，通常 *hbm.xml* 建议与对应的 POJO 类放置在一起。

接着就是建立 org.hibernate.cfg.Configuration 对象来读取设置文件，它是 *hibernate.cfg.xml* 中相关设置的具体代表对象，Configuration 对象建立之后就不会变动，如果您修改了设置文件的内容，则必须也另外再产生新的 Configuration 对象，这样新的变更才会被再次读取。

由于 Hibernate 默认会在主控台下产生许多日志消息，为了方便观看执行结果，可以在 *Classpath* 中放置一个 *log4j.properties* 文件，设置只有警示消息才加以显示，例如：

HibernateDemo — log4j.properties

```
log4j.rootLogger=WARN, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n
```

直接来看一个简单的程序，了解如何使用以上设计的程序进行数据的储存与查询：

HibernateDemo — HibernateDemo.java

```
package onlyfun.caterpillar;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateDemo {
    public static void main(String[] args) {
        // Configuration 负责管理 Hibernate 配置消息
        Configuration config =
            new Configuration().configure();
        // 根据 config 建立 SessionFactory
        // SessionFactory 将用于建立 Session
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        SessionFactory sessionFactory =
            config.buildSessionFactory();

        // 建立 DAO 对象
        IUserDAO userDAO = new UserDAO(sessionFactory);

        User user = new User();
        user.setName("caterpillar");
        user.setAge(new Integer(30));

        userDAO.insert(user);

        user = userDAO.find(new Integer(1));

        System.out.println("name: " + user.getName());
    }
}
```

在建立了 Configuration 对象之后，就可以使用它所包括的信息来建立 SessionFactory 对象，接着，在建立 UserDAO 时将 SessionFactory 的实例指定给它，然后就可以操作 UserDAO 对象来进行数据的储存了，一个执行的参考画面如图 6.4 所示。

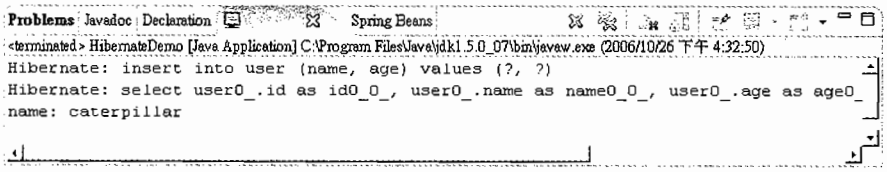


图 6.4 HibernateDemo 项目的执行结果

良葛格的话匣子<<<

有关于 Hibernate 更详尽的介绍，可以参考我网站上的 Hibernate 文件：

http://caterpillar.onlyfun.net/Gossip/HibernateGossip/ HibernateGossip.html

6.2 在 Spring 中整合 Hibernate

Spring 整合了对 Hibernate 的设置，并提供了 org.springframework.orm.hibernate3.HibernateTemplate 等类，让您在结合 Hibernate 使用时可以简化程序编写的流

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

程，并且以与 JDBC 相类似的使用模型提供使用 Hibernate 时的编程事务管理与声明事务管理。

6.2.1 SessionFactory 注入

Spring 可以与 Hibernate 结合使用，Hibernate 的连结、事务管理等是由建立 SessionFactory 开始的，SessionFactory 在应用程序中通常只需存在一个实例，因而 SessionFactory 底层的 DataSource 可以使用 Spring 的 IoC 注入，之后再把 SessionFactory 注入至相依赖的对象之中。

例如可以改写一下 6.1.3 节中的 HibernateDemo 项目，首先可删除 *hibernate.cfg.xml*，因为这部分可以由 Spring 在 Bean 定义文件中编写 DataSource 设置及依赖注入来取代，可以如下编写 Spring 的 Bean 定义文件：

SpringHibernateDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                 → datasource.DriverManagerDataSource">
       <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
       <property name="url"
                value="jdbc:mysql://localhost:3306/demo"/>
       <property name="username" value="caterpillar"/>
       <property name="password" value="123456"/>
    </bean>

    <bean id="sessionFactory"
         class="org.springframework.orm.
                 → hibernate3.LocalSessionFactoryBean"
         destroy-method="close">
       <property name="dataSource" ref="dataSource"/>
       <property name="mappingResources">
          <list>
             <value>onlyfun/caterpillar/User.hbm.xml</value>
          </list>
       </property>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
    </bean>

    <bean id="userDAO" class="onlyfun.caterpillar.UserDAO">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
</beans>
```

可以看到使用 Spring 整合 Hibernate 的好处，可以直接将 DataSource 注入至 org.springframework.orm.hibernate3.LocalSessionFactoryBean 中，至于 Hibernate 所需的相关设置，则可通过 LocalSessionFactoryBean 的相关属性来设置，像设置数据库名称、用户名称、密码等，LocalSessionFactoryBean 会建立 SessionFactory 的实例，并在执行依赖注入时将这个实例设置给 UserDAO。

Hibernate 的对象与关联表格的映射文件的位置与名称，指定在 "mappingResources" 属性中，如果自行提供 Hibernate 本身的设置文件（*hibernate.cfg.xml*），也可以使用 "configLocation" 属性来指定设置文件的位置，而这里则使用 "hibernateProperties" 属性在 Spring 的 Bean 定义文件中直接指定，可以借此减少对 XML 配置文件的管理。

良葛格的话匣子<<<

注意！在 Spring 2.0 中可以选择要整合 Hibernate 2 还是 Hibernate 3（Hibernate 3 在包名称上与 Hibernate 2 是不相容的），因而在 Spring 整合时会有两种版本的 API。org.springframework.orm.hibernate.LocalSessionFactoryBean 是在整合 Hibernate 2 时所使用的类，而 org.springframework.orm.hibernate3.LocalSessionFactoryBean 则是在整合 Hibernate 3 时使用的类，因为主要的类名称是相同的（例如 LocalSessionFactoryBean），所以编写时要注意包名称的设置是否正确。

项目中剩下的部分都不用修改，可以编写一个简单的测试程序来看看如何使用 Spring 整合 Hibernate，并进行简单的数据存取：

SpringHibernateDemo　　SpringHibernateDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.context.
            support.ClassPathXmlApplicationContext;

public class SpringHibernateDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "beans-config.xml");

        // 建立 DAO 对象
        IUserDAO userDAO =
            (IUserDAO) context.getBean("userDAO");

        User user = new User();
        user.setName("caterpillar");
        user.setAge(new Integer(30));

        userDAO.insert(user);

        user = userDAO.find(new Integer(1));

        System.out.println("name: " + user.getName());
    }
}
```

执行的结果基本上与 HibernateDemo 项目是相同的，可以参考图 6.4 的执行画面。

良葛格的话匣子<<<

在 SpringHibernateDemo 项目中，如果您不是选择加入 *spring.jar* 的话，则必须分别加入以下的文件至 *Classpath* 路径之中：*sping-core.jar*、*spring-beans.jar*、*spring-context.jar*、*spring-jdbc.jar*、*spring-dao.jar*、*spring-hibernate.jar*。

在映射文件的指定上，除了使用 "mappingResources" 属性设置之外，也可以使用 "mappingDirectoryLocations" 属性设置，一次性指定某个路径下的所有 *.hbm.xml* 文件，例如：

```
...
<bean id="sessionFactory"
     class="org.springframework.orm.hibernate3.
            → LocalSessionFactoryBean"
     destroy-method="close">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingDirectoryLocations">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <list>
            <value>classpath:/onlyfun/caterpillar</value>
        </list>
    </property>
    ...
</bean>
...
```

6.2.2 HibernateTemplate

Spring 的 Template-callback 机制在 Hibernate 的实现提供了 org.springframework.orm.ibernate3.HibernateTemplate 类与 org.springframework.orm.hibernate3.HibernateCallback 接口，一个使用的例子如下所示。

```
HibernateTemplate hibernateTemplate =
        new HibernateTemplate(sessionFactory);
...
hibernateTemplate.execute(new HibernateCallback() {
    public Object doInHibernate(
            Session session) throws HibernateException {
        return session.load(User.class);
    }
});
```

执行完毕之后，Session 对象会自动关闭。HibernateTemplate 上也提供有数个方便的方法，在执行时自动建立 HibernateCallback 对象，例如 load()、get()、save、delete() 等方法。可以改写 SpringHibernateDemo 项目中的 UserDAO 类，使用 HibernateTemplate 来简化程序的编写：

HibernateTemplateDemo — UserDAO.java

```
package onlyfun.caterpillar;

import org.hibernate.SessionFactory;
import org.springframework.orm.hibernate3.HibernateTemplate;

public class UserDAO implements IUserDAO {
    private HibernateTemplate hibernateTemplate;

    public void setSessionFactory(
            SessionFactory sessionFactory) {
        hibernateTemplate =
            new HibernateTemplate(sessionFactory);
    }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void insert(User user) {
        hibernateTemplate.save(user);
    }

    public User find(Integer id) {
        User user = (User) hibernateTemplate.get(User.class, id);

        return user;
    }
}
```

其他的文件并不需要修改，执行的结果也是相同的。

良葛格的话匣子<<<

您必须在 Classpath 中加入 spring-orm.jar 文件才可以执行 HibernateTemplateDemo 项目中的程序。

您可以继承 org.springframework.orm.hibernate3.support.HibernateDaoSupport 类来编写 UserDAO 类，这可以省去一些管理 SessionFactory、HibernateTemplate 资源的工作，只要注入 SessionFactory 的实例就可以了，例如将上面的 UserDAO 替代为以下的内容，则程序也可以执行：

```
package onlyfun.caterpillar;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

public class UserDAO extends HibernateDaoSupport implements IUserDAO {

    public void insert(User user) {
        getHibernateTemplate().save(user);
    }

    public User find(Integer id) {
        User user = (User) getHibernateTemplate().get(User.class, id);

        return user;
    }
}
```

注意到，在这里都是使用 HibernateTemplate 的 get() 方法，如果是使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

HibernateTemplate 的 load()方法，将会用到 Session 对象的 load()方法，这将会使用 Hibernate 3 默认的延迟加载功能，但 HibernateTemplate 执行完 load()方法之后，会直接关闭 Session，因而此时若再尝试取得 User 的属性，将会发生 LazyInitializationException。

解决的方法之一，是实现 HibernateCallback 接口，并将实例传入给 HibernateTemplate 的 execute()方法，例如：

```
...
   public User find(final Integer id) {
      User user = (User) hibernateTemplate.execute(
            new HibernateCallback() {
               public Object doInHibernate(Session session)
                     throws HibernateException, SQLException {
                  User user = (User) session.load(User.class, id);
                  Hibernate.initialize(user);
                  return user;
               }
            });

      return user;
   }
```

在 doInHibernate()方法的实现中，主动使用 Hibernate.initialize()方法初始化 User 对象的属性。另一个方法，就是直接在 Hibernate 的 *.hbm.xml* 中设置"lazy"属性为"false"，表示不使用延迟加载功能，例如修改 *User.hbm.xml*：

```
...
<hibernate-mapping>
   <class name="onlyfun.caterpillar.User" table="user" lazy="false">
      ...
   </class>
</hibernate-mapping>
```

6.2.3 HibernateTemplate 的 Lob 支持

在使用 Spring 搭配 Hibernate 时，可以简化对 Lob 类型的处理，只要在 SessionFactory 构造时指定 LobHandler 注入，并将 SessionFactory 注入给 HibernateTemplate，例如：

```
...
   <bean id="lobHandler" class="org.springframework.jdbc.
            → support.lob.DefaultLobHandler"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="sessionFactory" class="org.springframework.orm.
                → hibernate3.LocalSessionFactoryBean"
          destroy-method="close">
        <property name="dataSource" ref="dataSource"/>
        <property name="lobHandler" ref="lobHandler"/>
        <property name="mappingResources">
            <list>
                <value>onlyfun/caterpillar/User.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
    </bean>

    <bean id="hibernateTemplate" class="org.springframework.orm.
              → hibernate3.HibernateTemplate">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
...
```

在指定 LobHandler 时，对于 MySQL、DB2、MS SQL Server 和 Oracle 10g 使用 DefaultLobHandler 即可，而对于 Oracle 9i，则可以使用 OracleLobHandler。

接下来的操作与一般对 HibernateTemplate 的操作无异，例如您的数据库表格为：

```
CREATE TABLE user (
    id INT auto_increment PRIMARY Key,
    txt TEXT,
    image BLOB
);
```

Spring 的 ClobStringType 可以将 Clob 映射至 String，而 BlobByteArrayType 可以将 Blob 映射至 byte[]，可以设计一个 User 类，如下所示。

```
package onlyfun.caterpillar;

public class User {
    private Integer id;
    private String txt;
    private byte[] image;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public byte[] getImage() {
        return image;
    }
    public void setImage(byte[] image) {
        this.image = image;
    }
    public String getTxt() {
        return txt;
    }
    public void setTxt(String txt) {
        this.txt = txt;
    }
}
```

至于 *Use.hbm.xml*，则可以如下编写：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="onlyfun.caterpillar.User" table="user">
        <id name="id" column="id">
            <generator class="native"/>
        </id>
        <property name="txt" column="txt"/>
        <property name="image" column="image"/>
    </class>
</hibernate-mapping>
```

以下是个简单储存与读取 Lob 的程序片段示范：

```
...
    ApplicationContext context =
            new FileSystemXmlApplicationContext("beans-config.xml");

    InputStream is = new FileInputStream(
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                    new File("c:\\workspace\\wish.jpg"));
    byte[] b = new byte[is.available()];
    is.read(b);
    is.close();

    User user = new User();
    user.setTxt("long...long...text");
    user.setImage(b);

    HibernateTemplate hibernateTemplate =
        (HibernateTemplate) context.getBean("hibernateTemplate");
    hibernateTemplate.save(user);

    user = (User) hibernateTemplate.execute(new HibernateCallback() {
       public Object doInHibernate(Session session)
                    throws HibernateException, SQLException {
          User user = (User) session.load(User.class, new Integer(1));
          Hibernate.initialize(user);
          return user;
       }
    });

    System.out.println(user.getTxt());
    b = user.getImage();

    OutputStream os = new FileOutputStream(
                    new File("c:\\workspace\\wish_bak.jpg"));
    os.write(b);
    os.close();
...
```

6.2.4　Hibernate 编程事务管理

Hibernate 提供有自己的事务管理实现，然而在使用 Spring 整合 Hibernate 时，建议将事务管理交由 Spring 来负责，您可以使用编程式的事务管理，方法与第 5 章中直接使用 JDBC 进行编程式的事务管理类似，例如可将 HibernateTemplateDemo 项目中的 UserDAO 类加以改写，使之具有事务管理功能。

TransactionTemplateDemo　　UserDAO.java

```
package onlyfun.caterpillar;

import org.hibernate.SessionFactory;
import org.springframework.dao.DataAccessException;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.
        support.TransactionCallbackWithoutResult;
import org.springframework.transaction.support.TransactionTemplate;
import org.springframework.orm.
          hibernate3.HibernateTransactionManager;

public class UserDAO implements IUserDAO {
   private TransactionTemplate transactionTemplate;
   private HibernateTemplate hibernateTemplate;

   public void setSessionFactory(
          SessionFactory sessionFactory) {
      this.transactionTemplate =
          new TransactionTemplate(
                new HibernateTransactionManager(
                      sessionFactory));
      this.hibernateTemplate =
          new HibernateTemplate(sessionFactory);
   }

   public void insert(User user) {
      final User userData = user;

      // 设置传播行为
      transactionTemplate.setPropagationBehavior(
            TransactionDefinition.PROPAGATION_REQUIRED);

      transactionTemplate.execute(
        new TransactionCallbackWithoutResult() {
            protected void doInTransactionWithoutResult(
                  TransactionStatus status) {
               hibernateTemplate.save(userData);
            }
         });
   }

   public User find(Integer id) {
      User user = (User) hibernateTemplate.get(User.class, id);
      return user;
   }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

org.springframework.transaction.support.TransactionTemplate 类在建立时，需要一个实现 PlatformTransactionManager 的实例，这里使用 org.springframework.orm.hibernate3. ibernateTransactionManager 类来建立实例，TransactionTemplate 必须设置传播行为（Propagation Behavior）、并使用 Callback 对象来执行事务，在发生异常时将会撤消操作，如果需要细部控制何时撤消操作，可以在捕捉到异常之后，使用 TransactionStatus 的 setRollbackOnly()方法撤消操作，这些动作与第 5 章讲解 JDBC 事务管理时都是类似的，可以参考一下第 5 章有关事务管理的内容，或看看 Spring 参考手册来了解一些细节说明。

6.2.5　Hibernate 声明事务管理

Spring 对 Hibernate 提供声明事务管理，这与第 5 章中介绍过的 JDBC 声明事务管理类似，先来介绍一下比较简单的方式，直接从 HibernateTemplateDemo 项目进行改写，事实上不需要修改 UserDAO 类，而只要在定义文件上设置即可：

HibernateDeclarativeTransactionDemo　　UserDAO.java

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                → datasource.DriverManagerDataSource">
       <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
       <property name="url"
                value="jdbc:mysql://localhost:3306/demo"/>
       <property name="username" value="caterpillar"/>
       <property name="password" value="123456"/>
    </bean>

    <bean id="sessionFactory"
         class="org.springframework.orm.
                → hibernate3.LocalSessionFactoryBean"
         destroy-method="close">
       <property name="dataSource" ref="dataSource"/>
       <property name="mappingResources">
          <list>
             <value>onlyfun/caterpillar/User.hbm.xml</value>
          </list>
       </property>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
    </bean>

    <bean id="userDAO" class="onlyfun.caterpillar.UserDAO">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="transactionManager"
          class="org.springframework.orm.
                  → hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="userDAOProxy"
          class="org.springframework.transaction.
                  → interceptor.TransactionProxyFactoryBean">
        <property name="transactionManager"
                  ref="transactionManager"/>
        <property name="proxyInterfaces">
            <list>
                <value>onlyfun.caterpillar.IUserDAO</value>
            </list>
        </property>
        <property name="target" ref="userDAO"/>
        <property name="transactionAttributes">
            <props>
                <prop key="insert">PROPAGATION_REQUIRED</prop>
            </props>
        </property>
    </bean>

</beans>
```

主要的差别在于设置文件中建立了"transactionManager"与"userDAOProxy"，由于 Spring 所提供的是一致性的设置模型，所以设置的方式与第 5 章中介绍 JDBC 声明事务管理的设置方式是类似的，可以参考一下第 5 章有关于声明事务管理的说明。

来写个程序测试一下，这次取回 DAO 代理对象来进行操作，如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

HibernateDeclarativeTransactionDemo SpringHibernateDemo.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class SpringHibernateDemo {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "beans-config.xml");

        // 建立 DAO 对象
        IUserDAO userDAO =
            (IUserDAO) context.getBean("userDAOProxy");

        User user = new User();
        user.setName("cater");
        user.setAge(new Integer(30));

        userDAO.insert(user);

        user = userDAO.find(new Integer(1));

        System.out.println("name: " + user.getName());
    }
}
```

如果在执行 insert() 方法时发生了异常，则数据不会被新增至表格中，您可以自行在 UserDAO 类中尝试丢出个异常，以测试事务的功能是否正确执行。

良葛格的话匣子<<<

同样的，在 Spring 中对 Hibernate 进行声明事务管理时，使用了 Spring AOP 的功能，所以您必须在 *Classpath* 的路径设置中加入 *spring-aop.jar* 文件才可以执行项目中的程序。

在 5.3.6 节介绍过 Spring 2.0 基于 XML Schema 的声明事务管理，在这里若要使用，则只要修改 *beans-config.xml* 为以下设置方式：

HibernateDeclarativeTransactionDemo2 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">

  <bean id="dataSource"
       class="org.springframework.jdbc.
              → datasource.DriverManagerDataSource">
     <property name="driverClassName"
              value="com.mysql.jdbc.Driver"/>
     <property name="url"
              value="jdbc:mysql://localhost:3306/demo"/>
     <property name="username" value="caterpillar"/>
     <property name="password" value="123456"/>
  </bean>

  <bean id="sessionFactory"
       class="org.springframework.orm.
              → hibernate3.LocalSessionFactoryBean"
       destroy-method="close">
     <property name="dataSource" ref="dataSource"/>
     <property name="mappingResources">
        <list>
           <value>onlyfun/caterpillar/User.hbm.xml</value>
        </list>
     </property>
     <property name="hibernateProperties">
        <props>
           <prop key="hibernate.dialect">
              org.hibernate.dialect.MySQLDialect
           </prop>
        </props>
     </property>
  </bean>

  <bean id="userDAO" class="onlyfun.caterpillar.UserDAO">
     <property name="sessionFactory" ref="sessionFactory"/>
  </bean>

  <bean id="transactionManager"
       class="org.springframework.orm.
              → hibernate3.HibernateTransactionManager">
     <property name="sessionFactory" ref="sessionFactory"/>
  </bean>

  <tx:advice id="txAdvice"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="insert" propagation="REQUIRED"/>
            <tx:method name="find" read-only="true"/>
        </tx:attributes>
    </tx:advice>

    <aop:config>
        <aop:pointcut id="userDAOPointcut" expression=
            "execution(* onlyfun.caterpillar.IUserDAO.*(..))"/>
        <aop:advisor advice-ref="txAdvice"
                pointcut-ref="userDAOPointcut"/>
    </aop:config>

</beans>
```

由于不再于设置文件中设置代理对象，所以直接取得"userDAO"实例进行操作即可。您也可以使用基于 Annotation 的设置方式，例如改写 UserDAO，如下所示。

HibernateDeclarativeTransactionDemo3　　UserDAO.java

```
package onlyfun.caterpillar;

import org.hibernate.SessionFactory;
import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

public class UserDAO implements IUserDAO {
    private HibernateTemplate hibernateTemplate;

    public void setSessionFactory(
            SessionFactory sessionFactory) {
        hibernateTemplate =
            new HibernateTemplate(sessionFactory);
    }

    @Transactional(propagation = Propagation.REQUIRED)
    public void insert(User user) {
        hibernateTemplate.save(user);
    }

    @Transactional(readOnly=true)
    public User find(Integer id) {
        User user = (User) hibernateTemplate.get(User.class, id);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return user;
    }
}
```

接着在 *beans-config.xml* 中使用<tx:annotation-driven>并设置 TransactionManager 即可，例如：

HibernateDeclarativeTransactionDemo3 beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx-2.0.xsd">

    <bean id="dataSource"
         class="org.springframework.jdbc.
                → datasource.DriverManagerDataSource">
        <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
        <property name="url"
                value="jdbc:mysql://localhost:3306/demo"/>
        <property name="username" value="caterpillar"/>
        <property name="password" value="123456"/>
    </bean>

    <bean id="sessionFactory"
         class="org.springframework.orm.
                → hibernate3.LocalSessionFactoryBean"
         destroy-method="close">
        <property name="dataSource" ref="dataSource"/>
        <property name="mappingResources">
            <list>
                <value>onlyfun/caterpillar/User.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        </property>
    </bean>

    <bean id="userDAO" class="onlyfun.caterpillar.UserDAO">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="transactionManager"
         class="org.springframework.orm.
                → hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager"/>

</beans>
```

6.3　接下来的主题

对 Spring 于持久层的支持，暂且说明到这里，在 Spring 的参考手册中，还有更多有关于持久层方案的说明，建议您也可以参考一下。在下一个章节的开始，将介绍一下 Spring 于 Web 层所提供的解决方案，首先认识 Spring 所提供的 MVC 框架，使用它的好处是易于整合使用 Spring 的 IoC 容器功能。



图 6.5　Spring 整合并简化了 Hibernate 的使用方式

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第

Spring Web MVC 框架

章

Web MVC 架构是长久以来开发 Web 应用程序时，受多数开发人员所推荐的架构，现今有许多开放源代码（OpenSource）项目的 Web MVC 框架（Framework），Spring 也提供自己的 Web MVC 框架方案，希望能结合 Spring 的 IoC 容器功能，解决现存的 Web MVC 框架问题，使之在许多功能的整合上更为方便。

除了提供 Web MVC 框架之外，Spring 还提供了一个完善的 `Controller` 继承架构，让您可以依自己的需求使用适当的 `Controller` 类。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.1　Spring Web MVC 入门

在这节中，将从一个最简单的 Spring Web 应用程序，来了解 Web MVC 架构的运作方式，以及 Spring 的 Web 框架如何实现 Web MVC 架构。

7.1.1　使用 Eclipse Web Tools Platform

本章的 Web 应用程序，将使用 Eclipse Web Tools Platform（WTP）来进行开发，您可以在 *http://www.eclipse.org/webtools/*的“downloads”区下载“WebTools Platform; All-in-one”的版本。本章将使用 Tomcat 5.5 作为 Servlet 容器（Container），所以也请至 *http://tomcat.apache.org/* 下载 Tomcat 5.5。

良葛格的话匣子<<<

如果打算使用 Eclipse 的联机更新方式安装 WTP，则可以参考下列网址的影音教学：*http://eclipse.cdpa.nsysu.edu.tw/technology/phoenix/demos/install-wtp/install-wtp.html*。

在执行 WTP 之后，为了编写 Web 应用程序时，可以使用一些方便的工具视窗组合，请执行“Window/Open Perspective/Other...”，选择开启“J2EE Perspective”，如图 7.1 所示。

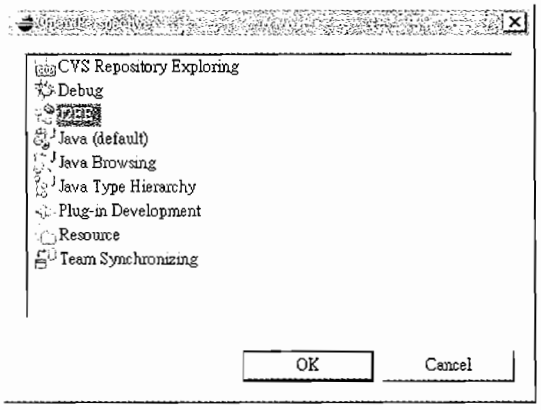


图 7.1　选择开启 J2EE Perspective

首先要设置 Server 执行环境，如此就可以在 WTP 中直接启动 Tomcat 容器，以测试在 WTP 中所编写的 Web 应用程序，请在 WTP 视窗下方选择“Servers”，并于视窗中单击鼠标右键，执行“New/Server”，如图 7.2 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

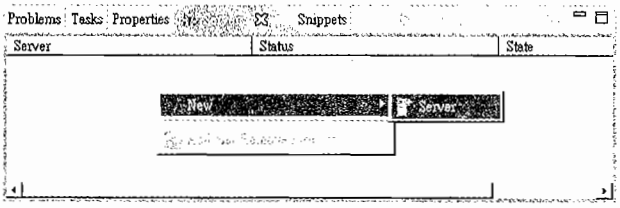


图 7.2 新增 Server 执行环境

因为本章将使用 Tomcat 5.5，所以接着在“New Server”视窗中选择“Apache”中的“Tomcat v5.5 Server”，如图 7.3 所示。

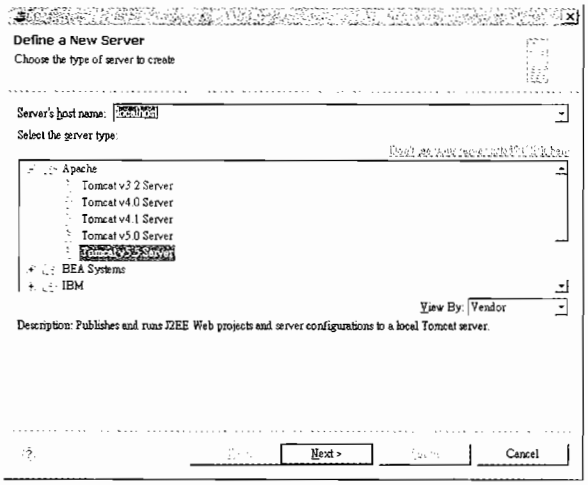


图 7.3 设置 Server 类型

点击“Next”按钮之后，请选择 Tomcat 安装文件夹，接着点击“Finish”按钮，即可完成 Server 执行环境的新增，完成后如图 7.4 所示。

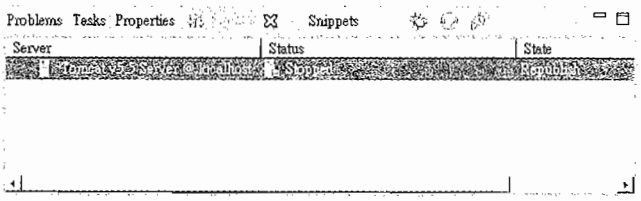


图 7.4 新增了 Server 执行环境

在 WTP 中要编写 Web 应用程序，可以在执行“File/New/Project...”新增项目时，选择“Web/Dynamic Web Project”项目，在点击“Next”按钮之后，输入项目名称，假设第一个

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Spring Web MVC 项目名称为 FirstSpringMVC，如图 7.5 所示。

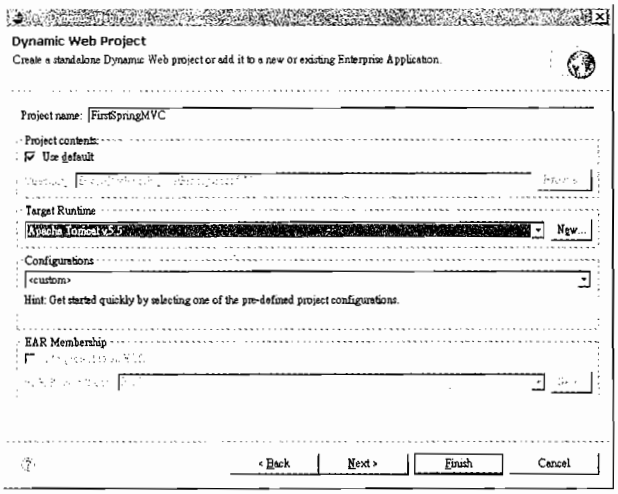


图 7.5　新增动态 Web 项目

点击“Finish”按钮，即可完成项目的新增。这次为了资源管理上的方便，直接使用 *spring.jar*，以及其相依赖的 *commons-logging.jar*，请将这两个 *jar* 放到 *WEB-INF/lib* 文件夹下，如图 7.6 所示。

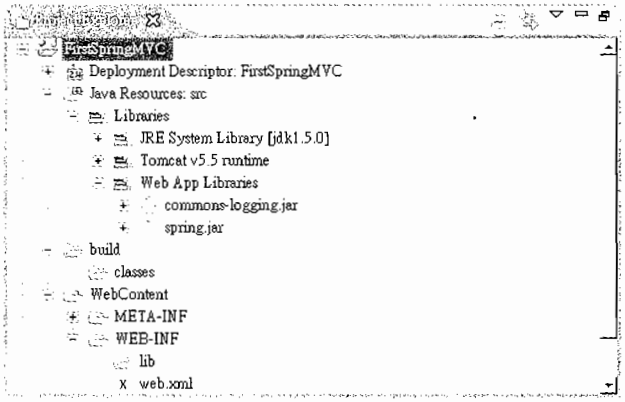


图 7.6　动态 Web 项目结构

在 WTP 中的 Java 源代码文件，或是必须放置在 Web 应用程序 *Classpath* 下的文件，可以新增在 *src* 目录之下，其他网页或 Web 相关资源，可以新增在 *WebContent* 目录之下，以后要导出（Export）为 WAR（Web Application Archive）文件时，WTP 会自动将相关文件组

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

织为 Web 应用程序的文件系统结构。

7.1.2 第一个 Spring Web MVC 程序

在 Web MVC 架构中，用户并不直接连接至所需的资源，而必须先连接至前端控制器（Front controller），由前端控制器判断用户的请求要分派（Dispatch）给哪一个控制对象（Controller）来处理请求，借此达到控制用户请求资源的目的。

在 Spring 的 Web MVC 框架中，担任前端控制器角色的是 `org.springframework.web.servlet.DispatcherServlet`，DispatcherServlet 负责将客户的请求分配给控制对象，所以使用 Spring Web MVC 的第一步，就是在 *web.xml* 中定义 DispatcherServlet：

FirstSpringMVC　　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  → http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/mvc-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>*.do</url-pattern>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    </servlet-mapping>
</web-app>
```

在 *web.xml* 中定义了一个 DispatcherServlet 的实例"dispatcherServlet"，从设置中可以看到，所有连接至 **.do* 结尾的请求都会由"dispatcherServlet"来处理，"contextConfigLocation" 初始参数用来设置 Bean 定义文件的位置与名称，如果不设置"contextConfigLocation" 初始参数，则 DispatcherServlet 默认会使用 Servlet 的名称为前置，读取“Servlet 名称-*servlet.xml*”作为其 Bean 定义文件，在上面的设置中则会读取 *mvc-config.xml* 中的定义。

您也可以定义多个 Bean 定义文件的来源，如：

```
...
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/mvc-config.xml,
            ➥ /WEB-INF/other-service.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
...
```

DispatcherServlet 负责分配请求至控制对象（Controller），在 Spring Web MVC 框架中，控制对象要实现 org.springframework.web.servlet.mvc.Controller 接口，Controller 接口有一个必须实现的 handleRequest()方法，其定义如下所示：

```
package org.springframework.web.servlet.mvc;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;

public interface Controller {
    ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws Exception;
}
```

当 Controller 收到 DispatcherServlet 分配的请求时，会执行 handleRequest()方

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

法来处理请求，处理完毕后回传一个 org.springframework.web.servlet.ModelAndView 的实例，ModelAndView 包括了要呈现在 View 层（例如 JSP 网页）的 Model 数据，以及其他 View 层的相关消息。

在第一个 Spring Web MVC 中，用户的请求将由一个 HelloController 对象来处理，其实现如下所示：

FirstSpringMVC　　HelloController.java

```
package onlyfun.caterpillar;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

public class HelloController implements Controller {
    private String viewPage;

    public ModelAndView handleRequest(HttpServletRequest req,
                                      HttpServletResponse res)
                                      throws Exception {
        String user = req.getParameter("user");
        return new ModelAndView(viewPage, "user", user);
    }

    public void setViewPage(String viewPage) {
        this.viewPage = viewPage;
    }
}
```

在这个 Controller 中，取得了来自用户的"user"请求参数，并设置在 ModelAndView 的实例中，在这个例子中，构造 ModelAndView 的第一个参数为要呈现的目标网页（或资源）路径，第二个参数是设置用来取得 Model 对象的"键（Key）"，而第三个参数是给 View 层呈现数据的 Model 对象。

良葛格的话匣子<<<

在 Web MVC 架构下，控制对象的作用是收集用户的请求，进行与 Web 层相关的动作，不应在控制对象中执行业务逻辑，也不应让 Servlet 相关的 API 侵入至业务层。这会让业务层的对象与 Servlet API 产生耦合，例如不能让 HttpServletRequest 对象直接设置至业务层对象之中。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

使用 Spring Web MVC 的好处是，Spring 的 Controller 在其 IoC 容器管理下，可以如同一般的 Bean 来加以管理，并利用其依赖注入来完成相关对象的注入，以这里的例子来说，您可以在 XML 文件中设置的 Controller 请求处理完毕之后，呈现数据的网页路径，来看一下 Bean 定义文件的内容，依 *web.xml* 中的设置，请在 *WEB-INF* 目录下建立 *mvc-config.xml* 文件：

FirstSpringMVC　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                ➞ view.InternalResourceViewResolver">
       <property name="prefix" value="/WEB-INF/jsp/"/>
       <property name="suffix" value=".jsp"/>
    </bean>

    <bean name="/hello.do"
         class="onlyfun.caterpillar.HelloController">
       <property name="viewPage" value="hello"/>
    </bean>

</beans>
```

实际上 DispatcherServlet 必须根据一个 HandlerMapping 对象，来决定请求由哪一个 Controller 处理，DispatcherServlet 默认是使用 org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping，也就是根据 Bean 在定义时的 "name" 属性及用户请求的 URL，决定使用哪一个 Controller 实例，例如在这个例子中，请求 /FirstSpringMVC/hello.do 时，DispatcherServlet 根据 "hello.do" 的名称，决定要使用 "name" 为 "hello.do" 的 Bean 实例，所以就是将请求交由 HelloController 的实例来处理。

当 Controller 传回 ModelAndView 后，DispatcherServlet 会交由 ViewResolver 对象来作 View 层的相关解析，因而您需要设置一个 ViewResolver 实例，在这个范例中将以 JSP 作为 View 层技术，所以使用 org.springframework.web.servlet.view.InternalResourceViewResolver，InternalResourceViewResolver 需要设置"viewClass"，默认是 org.springframework.web.servlet.view.InternalResourceView，这个类支持

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Servlet 技术的相关资源（如 JSP、Servlet）。

InternalResourceViewResolver 的 "prefix"、"suffix" 属性会与 ModelAndView 传回的路径信息相结合，例如若路径信息传回为 "hello" 字符串，则与以上例子的设置结合，实际的路径就是 */WEB-INF/jsp/hello.jsp*。

最后可以在 */WEB-INF/jsp/* 中写一个测试的 hello.jsp：

FirstSpringMVC　　hello.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>First Spring MVC</title>
</head>
<body>
    <h1>Hello, ${user}!!</h1>
</body>
</html>
```

在 ModelAndView 中设置的 Model 对象，经由 InternalResourceViewResolver 及 InternalResourceView 的解析，将被设置为 JSP 技术中可存取的属性（Attribute），因此可以使用 JSP 技术中的 Expression Language 来取得数据。依以上所编写的程序，如果您在请求 hello.do 时附带了 "user" 参数，则最后的 JSP 会出现 "user" 消息，可以执行 WTP 的“Run/Run As/Run On Server”指令，WTP 会自动启动 Tomcat 并启动 Web 浏览器，请将网址指向 *http://localhost:8080/FirstSpringMVC/hello.do?user=Justin*，一个执行的参考画面如图 7.7 所示。

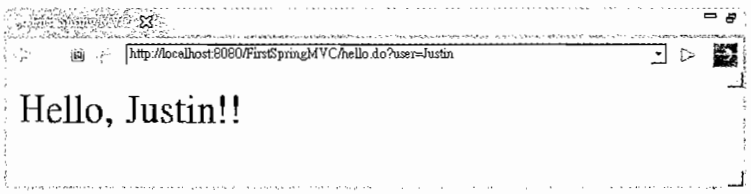


图 7.7　FirstSpringMVC 执行参考画面

7.1.3　WebApplicationContext

如果想要在自己所定义的 Servlet 类中使用 Spring 的容器功能，则也可以使用 org.springframework.web.context.ContextLoaderListener，例如在 *web.xml* 中使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

<listener> 标签加以定义：

```
...
<listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
...
```

ContextLoaderListener 默认会读取 applicationContext.xml，您可以指定自己的定义文件，只要在 <context-param> 中指定 "contextConfigLocation" 参数，例如：

```
...
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/beans-config.xml,
          → /WEB-INF/demo-service.xml</param-value>
</context-param>
...
```

接着可以在自定义的 Servlet 中使用 org.springframework.web.context.support.WebApplicationContextUtils，从 ServletContext 中取得 org.springframework.web.context.WebApplicationContext，例如：

```
WebApplicationContext ctx =
    WebApplicationContextUtils.
       getRequiredWebApplicationContext(
                    this.getServletContext());
```

WebApplicationContext 实现了 ApplicationContext 接口，是 Spring 专为 Servlet 的 Web 应用程序设计的 ApplicationContext 实现类，在取得 WebApplicationContext 之后，可以利用它来取得 Bean 定义文件中定义的 Bean 实例，例如：

```
Date date = (Date) ctx.getBean("dateBean");
```

在不支持 Listener 设置的容器上（例如 Servlet 2.2 以更早的版本），可以使用 org.springframework.web.context.ContextLoaderServlet 来取代上面的 ContextLoaderListener 的设置，例如：

```
...
<servlet>
    <servlet-name>contextLoader</servlet-name>
    <servlet-class>
      org.springframework.web.context.ContextLoaderServlet
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
...
```

综合以上的叙述，编写一个简单的范例来示范完整的配置方式，假设您编写了一个 Servlet 程序，作用是提供报时功能，如下所示：

WebApplicationContextDemo　　TimeServlet.java

```
package onlyfun.caterpillar;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.
                          support.WebApplicationContextUtils;

public class TimeServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
                    throws ServletException, IOException {
        WebApplicationContext ctx =
                WebApplicationContextUtils.
                    getRequiredWebApplicationContext(
                        this.getServletContext());

        PrintWriter out = res.getWriter();
        out.println(ctx.getBean("dateBean"));
    }
}
```

在这个 Servlet 中使用了 WebApplicationContext 来尝试取得"dateBean"，可以在 *web.xml* 中定义 ContextLoaderListener 与 Bean 定义文件的位置，例如：

WebApplicationContextDemo　　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  ➞ http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
    <session-config>
        <session-timeout>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            30
        </session-timeout>
    </session-config>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/beans-config.xml</param-value>
    </context-param>

    <listener>
        <listener-class>
         org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>

    <servlet>
        <servlet-name>time</servlet-name>
        <servlet-class>
            onlyfun.caterpillar.TimeServlet
        </servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>time</servlet-name>
        <url-pattern>/time.do</url-pattern>
    </servlet-mapping>
</web-app>
```

在 <context-param> 的 "contextConfigLocation" 属性中，设置了 Bean 定义文件的位置与名称，Bean 定义文件的内容如下所示：

WebApplicationContextDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="dateBean"
          class="java.util.Date" scope="singleton"/>

</beans>
```

可以使用浏览器连接至 TimeServlet，网页会显示连接的时间，一个执行的画面如图 7.8 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

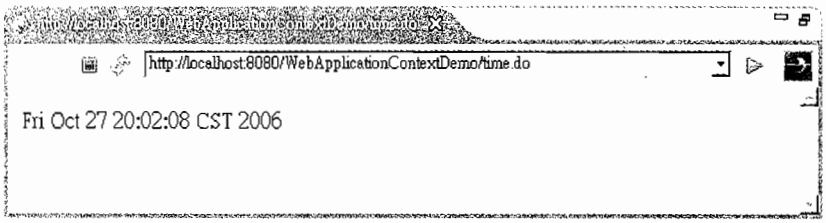


图 7.8 · WebApplicationContextDemo 的执行参考画面

7.1.4 Handler Mapping

当客户的请求来到时，DispatcherServlet 会根据 Handler mapping 对象决定，如何将请求分配至对应的 Controller。对于 Web 应用程序来说，通常就是以 URL 路径对应至 Controller。

DispatcherServlVet 默认的 Handler mapping 是 org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping，所以即使在 Bean 定义文件中没有明确声明 BeanNameUrlHandlerMapping，也会使用 BeanNameUrlHandlerMapping 依照 Bean 定义的 "name" 属性之 URL，来决定使用哪一个 Controller，例如：

```
...
   <bean id="viewResolver"
        class="org.springframework.web.servlet.
               → view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
   </bean>

   <bean name="/hello.do"
         class="onlyfun.caterpillar.HelloController">
      <property name="viewPage" value="hello"/>
   </bean>
...
```

在上面的设置中，用户对 */hello.do* 路径的请求，都会由 DispatchServlet 交由 HelloController 对象处理。

BeanNameUrlHandlerMapping 是简单的 Handler mapping，仅适用于小型的应用程序，因为它使 URL 与 Controller 的类名称直接建立了关系。在 Spring 的 Handler mapping 中最常使用的是 org.springframework.web.servlet.handler.SimpleUrlHandlerMapping，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

一个使用的例子如下所示：

```
...
    <bean id="urlHandlerMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/hello.do">helloController</prop>
                <prop key="/welcome.do">welcomeController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="helloController"
          class="onlyfun.caterpillar.HelloController">
        <property name="viewPage" value="hello"/>
    </bean>

    <bean id="welcomeController"
         class="onlyfun.caterpillar.WelcomeController">
        ...
    </bean>
...
```

在 "mappings" 属性的设置中，<prop> 的 "key" 是请求的 URL，而在 <prop> 与 </prop> 间设置实际上处理请求的 Bean 名称，<prop> 的 "key" 设置还可以使用 Wildcard，例如：

```
...
    <bean id="urlHandlerMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/hello*.do">helloController</prop>
                <prop key="/welcome.do">welcomeController</prop>
            </props>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        </property>
    </bean>
....
```

在上面的设置中，以 hello 开头及 .do 结尾的 URL 请求，都会交给 Bean 定义文件中的"helloController"来处理。

若定义 Controller 名称时遵守命名规范，则可使用 Spring 2.0 新增的 org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping，例如可以改写 FirstSpringMVC 的 *mvc-config.xml* 如下：

```
...
    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="controllerClassNameHandlerMapping"
        class="org.springframework.web.servlet.mvc.
                → support.ControllerClassNameHandlerMapping"/>

    <bean name="helloController"
         class="onlyfun.caterpillar.HelloController">
        <property name="viewPage" value="hello"/>
    </bean>
...
```

ControllerClassNameHandlerMapping 会自动将 hello*的请求与 Controller 类名称比对，例如 hello.do 的请求将会尝试找寻 HelloController 实例来处理，同样的，若是 welcome* 的请求， ControllerClassNameHandlerMapping 则会尝试找寻 WelcomeController 实例来处理。

7.1.5 Handler Interceptor

如果在 Controller 执行的前后，或者是在 View 绘制之后打算作一些记录或拦截请求等动作，可以实现 org.springframework.web.servlet.HandlerInterceptor 接口，在这个接口中规范了 3 个方法，其定义如下所示：

```
package org.springframework.web.servlet;

import javax.servlet.http.HttpServletRequest;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import javax.servlet.http.HttpServletResponse;

public interface HandlerInterceptor {
    boolean preHandle(HttpServletRequest request,
              HttpServletResponse response,
                  Object handler) throws Exception;
    void postHandle(HttpServletRequest request,
              HttpServletResponse response,
              Object handler,
              ModelAndView modelAndView) throws Exception;
    void afterCompletion(HttpServletRequest request,
              HttpServletResponse response,
              Object handler,
              Exception ex) throws Exception;
}
```

preHandler() 会在 Controller 处理请求之前被执行，传回的 boolean 值决定是否让 Handler Interceptor 或是 Controller 来处理请求，如果传回 false，则接下来的 Interceptor 或 Controller 就不处理请求，postHandler() 则会在 Controller 处理完请求之后被执行，afterCompletion() 方法也会在 View 绘制完成之后被执行。

您可以继承 org.springframework.web.servlet.handler.HandlerInterceptorAdapter，它实现了 HandlerInterceptor 接口，您只要针对必要的方法重新定义，例如可以修改 FirstSpringMVC 项目，在当中实现 LoggingInterceptor，在请求被控制对象处理的前、后来作记录：

HandlerInterceptorDemo　　LoggingInterceptor.java

```
package onlyfun.caterpillar;

import java.util.logging.Logger;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.
               handler.HandlerInterceptorAdapter;

public class LoggingInterceptor
       extends HandlerInterceptorAdapter {
    private Logger logger =
           Logger.getLogger(this.getClass().getName());

    public boolean preHandle(HttpServletRequest request,
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                HttpServletResponse response,
                Object handler) throws Exception {
        logger.info(
                handler.getClass().getName() + " 开始执行...");
        return true;
    }

    public void postHandle(HttpServletRequest request,
                HttpServletResponse response, Object handler,
                ModelAndView modelAndView) throws Exception {
        logger.info(
                handler.getClass().getName() + " 执行完毕...");
    }

    public void afterCompletion(HttpServletRequest request,
                HttpServletResponse response, Object handler,
                Exception ex) throws Exception {
        logger.info("请求处理完毕...");
    }
}
```

要使用这一个 HandlerInterceptor，必须在 Bean 定义文件中进行定义，例如：

HandlerInterceptorDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="loggingInterceptor"
         class="onlyfun.caterpillar.LoggingInterceptor"/>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                ➥ view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="urlHandlerMapping"
         class="org.springframework.web.servlet.
                ➥ handler.SimpleUrlHandlerMapping">
        <property name="interceptors">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            <list>
                <ref bean="loggingInterceptor"/>
            </list>
        </property>
        <property name="mappings">
            <props>
                <prop key="/hello.do">helloController</prop>
            </props>
        </property>
    </bean>

    <bean id="helloController"
          class="onlyfun.caterpillar.HelloController">
        <property name="viewPage" value="hello"/>
    </bean>

</beans>
```

注意所使用的 UrlHandlerMapping 是 SimpleUrlHandlerMapping；当通过 DispatcherServlet 请求时，在请求被处理的前后都会进行记录，在控制台（Console）所看到的信息如图 7.9 所示。

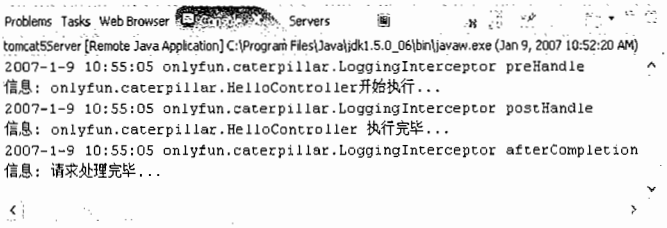


图 7.9　HandlerInterceptorDemo 的执行参考画面

7.1.6　Controller 继承架构

当用户送出请求之后，实际上处理请求的是 Controller，您可以实现 Controller 接口来处理请求，在 Spring 中，Controller 是 Spring IoC 容器管理的一个 Bean 实例。Spring 提供丰富的 Controller 继承架构，可以根据实际上的需求来继承某个类，以完成所需的 Controller 处理。

图 7.10 列出了 Spring 中 Controller 的继承架构。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

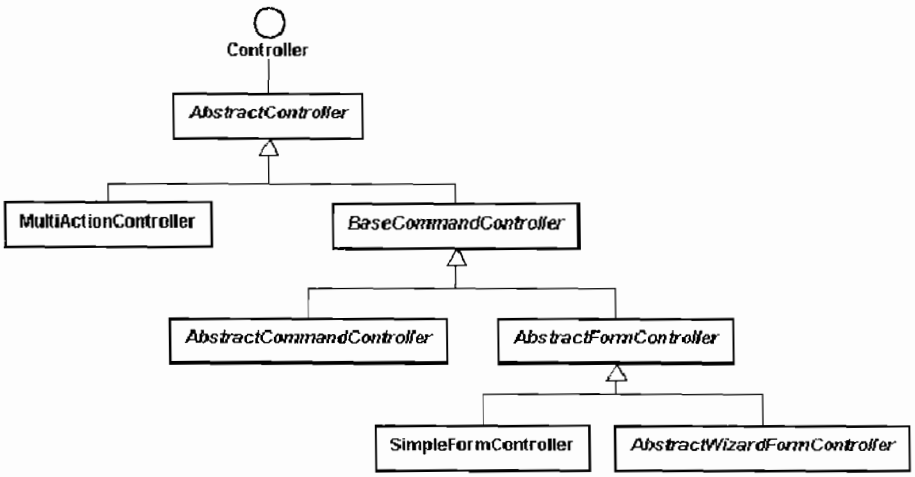


图 7.10 Spring 中 Controller 继承架构

以下分别简介 Controller 相关类之作用。

- AbstractController

AbstractController 实现了 Controller 接口，并继承了 WebContentGenerator，提供了 Session 缓存与同步化（synchronized）的处理。您可以直接继承 AbstractController，并重新定义 handleRequestInternal()方法来处理请求，例如：

```
public class SomeController extends AbstractController {
   public ModelAndView handleRequestInternal(
            HttpServletRequest request,
            HttpServletResponse response) throws Exception {
      ....
      return new ModelAndView("view", "modelName", model);
   }
   ..
}
```

- MultiActionController

如果为每一个请求编写一个 Controller 类，在程序规模增大时，您的 Controller 类将会迅速增加。可以继承或直接使用 MultiActionController 类，将数个相关的请求处理集中于同一个对象，而不用每一个请求编写一个控制对象。

- BaseCommandController、AbstractCommandController

如果需要将请求参数值撷取至一个 Command 对象中加以管理，则可以使用 Base-

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

CommandController 类的子类。在 Spring 中，Command 是一个 JavaBean，BaseCommandController 负责将请求参数撷取并设置至 Command 对象中，AbstractCommandController 则定义了后续的处理流程，通常会继承 AbstractCommandController 来定义 CommandController，一个例子如下：

```
public class SomeCommandController
                extends AbstractCommandController {
   public SomeCommandController() {
      setCommandClass(YourCommand.class);
   }

   public ModelAndView handle(HttpServletRequest request,
                HttpServletResponse response,
                Object command,
                BindException exception) throws Exception {
      YourCommand your = (YourCommand) command;
      ...
      return new ModelAndView(..);
   }
   ..
}
```

- AbstractFormController

对于表单处理，AbstractFormController 中定义了处理请求参数，以及表单提交成功或失败时所要进行的页面流程。

- SimpleFormController

对于简单的表单处理，可以继承 SimpleFormController 类，可以重新定义 doSubmitAction() 方法，如果要自己处理 ModelAndView，则可以重新定义 onSubmit() 方法。

- AbstractWizardFormController

对于需要数个表单画面来完成数据收集的网页来说，可以继承 AbstractWizardFormController，您可以像桌面应用程序的向导（Wizard）功能一样来制作表单。

在这里先对于以上的 Controller 作个简单的介绍，每一个 Controller 类在下一节中都会有详细的说明，以下将继续说明 Spring Web MVC 的主要 API。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.1.7 ModelAndView

ModelAndView 类代表了在 Spring Web MVC 程序中，呈现画面时所使用的 Model 数据对象与 View 数据对象，由于 Java 程序的方法中一次只能回传一个对象，所以 ModelAndView 的作用是封装这两个对象，方便您同时传回 Model 与 View。

最简单的 ModelAndView 是只包括 View 的名称，之后 View 名称被 View resolver，也就是被实现 org.springframework.web.servlet.View 接口的实例解析，例如 InternalResourceView 或 JstlView 等，最简单的 ModelAndView 构造方式如下：

```
ModelAndView(String viewName)
```

如果要返回呈现画面时所需的 Model 数据对象，则可以使用 Map 对象来收集数据，然后构造 ModelAndView 作为构造时的参数，可以使用下面这个版本的 ModelAndView 构造方法：

```
ModelAndView(String viewName, Map model)
```

在 Map 对象中设置“键（Key）”与“值（Value）”，之后可以在要呈现的画面中取出并加以显示（例如在 JSP 网页中），如果您要传回一个 Model 数据对象并指定 Model 的名称，则可以使用下面这个 ModelAndView 版本：

```
ModelAndView(String viewName, String modelName, Object modelObject)
```

通过 modelName，您可以在 View 的实现类中取出 Model 数据对象，并根据 View 所使用的技术来加以显示（可能是 JSP 网页、Pdf 等呈现技术）。

ModelAndView 类也可使用 View 接口的对象来作为构造时的参数，以下是 3 个可用的构造方法：

```
ModelAndView(View view)
ModelAndView(View view, Map model)
ModelAndView(View view, String modelName, Object modelObject)
```

一个实现 View 的实例是 org.springframework.web.servlet.view.RedirectView，ModelAndView 默认的是使用转发（Forward）方式来完成请求的结果画面，若使用 RedirectView 的话，则会使用重新导向（Redirect）将请求重导至指定的结果画面位置，以呈现请求的结果，例如：

```
...
public ModelAndView handleRequest(....) ... {
    ...
    return new ModelAndView(
                    new RedirectView(this.getViewPage()));
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}
...
```

在这里，viewPage 的设置必须从服务器网页的根目录开始，而不是从 Web 应用程序的根目录，所以 getViewPage() 传回的地址必须是像 */FirstSpring- MVC/pages/index.htm* 这样的地址，其中 *FirstSpringMVC* 是 Web 应用程序目录。

若使用转发（Forward）方式，地址栏上并不会出现被转发的地址，而且转发将在 Web 应用程序之内进行，可以访问 Web 应用程序所设置的内部目录，如 *WEB-INF* 目录，因此可以将一些要控制存取的资源放到 *WEB-INF* 下，如此用户就无法直接请求这些资源了，而必须通过 DispatcherServlet 与 Controller 的处理才可以取得这些资源。转发方式只能在 Web 应用程序中进行，不能指定至其他的 Web 应用程序地址。

若使用重新导向（Redirect），Web 应用程序会要求客户端浏览器重新发出请求地址，客户端也会重新连接至所指定的地址，因此浏览器的地址栏上会出现重新导向的信息，重新导向后的请求是由浏览器发出，所以不能访问 Web 应用程序中的隐藏目录，如 *WEB-INF*，重新导向是指由浏览器重新要求一个网页指定至其他的 Web 应用程序地址。

在 Spring 2.0 中，ModelAndView 新增了 addObject() 方法，您可以仅加入 Model 对象，addObject() 方法会自动产生一个“键”，例如：

```
...
   public ModelAndView handleRequest(HttpServletRequest req,
                                     HttpServletResponse res)
                                     throws Exception {

      User user = new User();
      ...
      ModelAndView modelAndView = new ModelAndView(viewPage);
      modelAndView.addObject(user);
      return modelAndView;
   }
...
```

如果设置给 addObject() 的是 onlyfun.caterpillar.User 实例，则会自动产生“键”（"user"），也就是说按照类或接口名称来产生名称，如果加入 null，则会发生 IllegalArgumentException。

Spring 2.0 的 ModelAndView 还可以不指定 View，由 RequestToViewNameTranslator 根据请求来提供 View 名称，例如可以在设置文件中如下编写：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="controllerClassNameHandlerMapping"
         class="org.springframework.web.servlet.mvc.
                → support.ControllerClassNameHandlerMapping"/>

    <bean id="viewNameTranslator"
         class="org.springframework.web.servlet.
                → view.DefaultRequestToViewNameTranslator"/>

    <bean name="helloController"
         class="onlyfun.caterpillar.HelloController">
        <property name="viewPage" value="hello"/>
    </bean>
...
```

在 HelloController 中实例化 ModelAndView 时，若不指定 View，例如：

```
...
public ModelAndView handleRequest(HttpServletRequest req,
                                  HttpServletResponse res)
                                  throws Exception {
    ModelAndView modelAndView = new ModelAndView();
    ...
    return modelAndView;
}
...
```

由于设置了 ControllerClassNameHandlerMapping，所以如果请求 hello.do，则会让 HelloController 来处理请求，由于设置了 RequestToViewNameTranslator，虽然 handleRequest()产生 ModelAndView 时并没有指定 View，但会由 *hello.jsp* 来处理（结合了 InternalResourceViewResolver 的前置与后置设置）。

7.1.8 View Resolver

DispatcherServlet 根据传回的 ModelAndView 实例来解析 View 名称，并处理呈现于画面的 Model 对象。View 名称的解析是委托给实现 org.springframework.web.servlet.ViewResolver 接口的实例，ViewResolver 接口定义如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
package org.springframework.web.servlet;

import java.util.Locale;

public interface ViewResolver {
   View resolveViewName(String, Locale locale)
                           throws ServletException;
}
```

ViewResolver 的一个实例是 InternalResourceViewResolver，您可以在 viewResolver 中设置 "prefix" 与 "suffix" 属性，分别设置路径的前置与后置文字，这样 viewPage 属性就可以只打路径的主要区别名称，直接给个设置的示范：

```
...
<bean id="viewResolver"
     class="org.springframework.web.servlet.
            → view.InternalResourceViewResolver">
   <property name="prefix" value="/WEB-INF/jsp/"/>
   <property name="suffix" value=".jsp"/>
</bean>
...
```

ViewResolve 在名称解析完后，实际的 View 绘制与 Model 对象的转换处理是交给实现 org.springframework.web.servlet.View 接口的实例，View 接口定义如下：

```
package org.springframework.web.servlet;

import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public interface View {
   void render(Map model, HttpServletRequest request,
         HttpServletResponse response) throws Exception;
}
```

例如 InternalResourceViewResolver 默认的 "viewClass" 是 InternalResourceView，也可以设置 "viewClass" 属性为其他的 View 层技术相关类，例如设置为 org.springframework.web.servlet.view.JstlView，另外还有 TilesView、VelocityView 等实现类，可适用不同的 View 技术。

您可以使用 org.springframework.web.servlet.view.BeanNameViewResolver，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
...
<bean id="viewResolver"
     class="org.springframework.web.servlet.
            → view.BeanNameViewResolver"/>

<bean id="hello"
     class="org.springframework.web.servlet.view.JstlView">
   <property name="url" value="/WEB-INF/jsp/hello.jsp"/>
</bean>
...
```

在以上的设置中，当 ModelAndView 返回的 view 名称为 "hello" 时，会去寻找 "id" 属性为"hello"的 View Bean。

可以将 ViewResolver 的定义独立出来放至一个属性文件中，这可以使用 org.springframework.web.servlet.view.ResourceBundleViewResolver 来达成，好处是可以根据实际的 URL 设置不同的表示层技术，例如：

```
...
<bean id="viewResolver"
     class="org.springframework.web.servlet.
            → view.ResourceBundleViewResolver">
   <property name="basename" value="views"/>
</bean>
...
```

在 ResourceBundleViewResolver 的 "basename" 属性设置了 "views" 名称（"views" 也是 "basename" 的默认值），这将使用 views.properties 来设置表示层技术的 "viewClass" 与 "url"，例如一个 views.properties 的编写范例如下所示：

```
hello.class=org.springframework.web.servlet.
               → view.InternalResourceView
hello.url=/WEB-INF/jsp/hello.jsp

welcome.class=org.springframework.web.servlet.view.JstlView
welcome.url=/WEB-INF/jsp/welcome.jsp

other.class=org.springframework.web.servlet.view.VelocityView
other.url=/WEB-INF/vm/other.vm
```

在传回的 ModelAndView 实例中的 view 名称若为 "hello"，则会寻找 *hello.class* 及 *hello.url* 来得到对应的 ViewClass 实例及指定的 URL，您可以为不同的国家或地区设置不同的 View，在 "basename" 上设置为 "views"，表示默认会读取 views.properties，如果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

语系设置是 en_US，可以为它准备 views_en_US.properties，如果语系设置是 zh_TW，可以为它准备 views_zh_TW.properties，而对应的文件中则可以设置与区域设置 Locale 相关的 View。

也可以使用 org.springframework.web.servlet.view.XmlViewResolver，一个设置的范例如下所示：

```
...
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="location" value="/WEB-INF/demo-views.xml"/>
</bean>
...
```

在上面的设置中，"location" 属性默认是 *views.xml*，这里设置为 *demo-views.xml*，表示到 *demo-views.xml* 中寻找 View 的 Bean 定义，例如在 *demo-views.xml* 中这么定义：

```
...
<bean id="hello"
      class="org.springframework.web.servlet.view.JstlView">
    <property name="url" value="/WEB-INF/jsp/hello.jsp"/>
</bean>
...
```

其运作的机制与设置 BeanNameViewResolver 时是类似的，当回传的 ModelAndView 实例中的 view 名称为 "hello" 时，则会去寻找 Bean 定义中的 "id" 为 "hello" 的 View 设置。

7.1.9　Exception Resolver

如果 JSP 网页或 Servlet 在运行过程中丢出了异常，而您想捕捉这个异常，除了使用容器与 JSP 网页中有关异常的捕捉设置方法外（例如设置 *web.xml* 的 <error-page> 标签，或是设置 JSP 网页 "page" 指令元素的 "errorPage" 属性），也可以在 DispatcherServlet 的定义文件中设置处理错误网页，即设置一个 ExceptionResolver 的 Bean 实例，例如使用 org.springframework.web.servlet.handler.SimpleMappingExceptionResolver，一个设置的例子如下所示：

```
...
<bean id="viewResolver"
      class="org.springframework.web.servlet.
            → view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <property name="suffix" value=".jsp"/>
</bean>

<bean id="exceptionResolver"
     class="org.springframework.web.servlet.
          → handler.SimpleMappingExceptionResolver">
    <property name="exceptionMappings">
        <props>
            <prop key="java.sql.SQLException">sqlexception</prop>
            <prop key="java.sql.IOException">ioexception</prop>
        </props>
    </property>
</bean>
...
```

DispatcherServlet 会自动应用 ExceptionResolver，并在异常发生时使用指定的网页来显示错误消息，例如在上面的设置中，只要发生了 SQLException，就会连接至 */WEB-INF/jsp/sqlexception.jsp*，而发生了 IOException，就会连接至 */WEB-INF/jsp/ioexception.jsp*。

7.2 Controller 实现类

Spring 提供实现 Controller 接口的相关类，可以依需求来制作所需的 Controller 对象，无论是简单的请求参数收集、表单网页的请求处理，甚至是向导（Wizard）网页的设计等，在 Spring 中都有对应的类可以应用。

7.2.1 AbstractController

org.springframework.web.servlet.mvc.AbstractController 类是 Controller 接口的简单实现，使用 Template Method 模式实现了用户请求处理的流程，包括了对缓存标头（Caching header）的处理、检验对请求方法（GET、POST）的支持、Session 的取得与同步化（synchronized）等，如果需要对这些议题作处理，则可以使用 AbstractController 类。

AbstractController 在 handleRequest() 方法中被 DispatcherServlet 开始执行的工作流程如下所示：

1. DispatcherServlet 执行 handleRequest() 方法。
2. 根据 "supportedMethods" 的设置来检验支持的请求方法，如果方法不支持则丢出 ServletException。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

3. 根据 "requireSession" 的设置决定请求是否需要使用 Session，尝试取得 Session，如果没有取得 Session 则丢出 ServletException。
4. 根据 "cacheSeconds" 的设置决定是否设置缓存标头（Caching header）。
5. 调用 handleRequestInternal() 方法，根据 "synchronizeOnSession" 决定是否对 Session 进行同步化（synchronized）。

对于 AbstractController 类的工作流程，只要有个基本的了解就可以了，可以直接继承 AbstractController 类，并重新定义 handleRequestInternal()方法，例如：

```
public class SomeController extends AbstractController {
    protected ModelAndView handleRequestInternal(
            HttpServletRequest request,
            HttpServletResponse response) throws Exception {
        ....
        return new ModelAndView("view", "modelName", model);
    }
    ..
}
```

若使用 AbstractController 类，则可以直接操作已经定义好的一些方法，例如可以调用 setSupportedMethods()方法，设置所允许的请求方式，例如若 Controller 实现时按如下编写，则只支持 GET、POST、HEAD 的请求：

```
package onlyfun.caterpillar;
...
public class HelloController extends AbstractController {
    ...
    public HelloController() {
        this.setSupportedMethods(
                new String[] {"GET", "POST", "HEAD"});
    }

    protected ModelAndView handleRequestInternal(
                HttpServletRequest req, HttpServletResponse res)
                                    throws Exception {
        ...
    }
}
```

如果以不允许的方式请求，Controller 在处理的时候则会丢出 org.springframework.web.servlet.support.RequestMethodNotSupportedException 的异常。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.2.2 MultiActionController 与 ParameterMethodNameResolver

如果应用程序的规模越来越大，若为每一个请求的处理编写一个 Controller 类，很快的，您会发现 Web 应用程序中有越来越多的 Controller 类，而事实上当中的一些职责是可以组织在一个类的，例如与数据库操作相关的 list()、add()、delete() 等方法，就可以组织在一个类里。

您可以使用 AbstractController 类的一个子类：org.springframework.web.servlet.mvc.multiaction.MultiActionController。它可以让您在一个 Controller 类中定义多个方法，并根据用户的请求来执行当中的某个方法。

要使用 MultiActionController，您要配合一个 org.springframework.web.servlet.mvc.multiaction.MethodNameResolver 的实例，MultiActionController 默认使用的 MethodNameResolver 类是 org.springframework.web.servlet.mvc.multiaction.InternalPathMethodNameResolver，这个类可以根据所给的网址中，最后请求的文件名称来判断执行 Controller 中的哪一个方法，例如 /test.do 的请求会执行 test(HttpServletRequest, HttpServletResponse) 方法。但通常不会使用 InternalPathMethodNameResolver，因为这就失去了使用 MultiActionController 的一些优点，像是依用户的请求参数来决定所要执行的方法。

通常在使用 MultiActionController 时会搭配 org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver 或是 org.springframework.web.servlet.mvc.multiaction.PropertiesMethodNameResolver，两个类都是根据一个请求参数，以参数决定要执行 Controller 类中的哪一个方法，这边先以 ParameterMethodNameResolver 为例，可以在 Bean 定义文件中这么编写：

MultiActionDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                 → handler.SimpleUrlHandlerMapping">
       <property name="mappings">
          <props>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            <prop key="/book.do">bookController</prop>
        </props>
    </property>
</bean>

<bean id="viewResolver"
     class="org.springframework.web.servlet.
            → view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean>

<bean id="paraMethodResolver"
     class="org.springframework.web.servlet.mvc.
            → multiaction.ParameterMethodNameResolver">
    <property name="paramName" value="action"/>
    <property name="defaultMethodName" value="list"/>
</bean>

<bean id="bookController"
     class="onlyfun.caterpillar.BookController">
    <property name="methodNameResolver"
            ref="paraMethodResolver"/>
    <property name="testPage" value="test"/>
</bean>

</beans>
```

在"paraMethodResolver"的 Bean 定义中，使用 "paramName" 属性设置在 HTTP 请求中使用 "action" 参数指定所要执行的方法，若是请求中没有指定 "action" 参数，则会使用 "defaultMethodName" 属性所设置的方法，这里设置的是 list() 方法。

BookController 继承 MultiActionController 类，它定义在 "action" 参数指定下所要执行的方法，例如：

MultiActionDemo　　BookController.java

```
package onlyfun.caterpillar;

import javax.servlet.http.*;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.
            multiaction.MultiActionController;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class BookController extends MultiActionController {
    private String testPage;

    public ModelAndView list(HttpServletRequest req,
                      HttpServletResponse res) {
        return new ModelAndView(
                    this.getTestPage(),"executed", "list");
    }

    public ModelAndView add(HttpServletRequest req,
                       HttpServletResponse res) {
        return new ModelAndView(
                    this.getTestPage(),"executed", "add");
    }

    public ModelAndView delete(HttpServletRequest req,
                         HttpServletResponse res) {
        return new ModelAndView(
                   this.getTestPage(),"executed", "delete");
    }

    public String getTestPage() {
        return testPage;
    }

    public void setTestPage(String testPage) {
        this.testPage = testPage;
    }
}
```

注意到在定义的方法中，必须包括 HttpServletRequest 与 HttpServletResponse 的参数，也可以使用带有第三个参数 HttpSession 的版本。

以上所定义的 BookController 类只是一个简单的范例，用于测试 MultiActionController 的运作，它所作的处理只是设置一个字符串，表示目前所执行的方法名称，可以设计一个测试用的 JSP 网页，用以显示用户所请求的方法，例如：

MultiActionDemo　　test.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>MultiAction Test Page</title>
</head>
<body>
   <H1> "${executed}" method is executed.</H1>
</body>
</html>
```

来实际看个发出请求的例子，可以在网址中附上请求参数，一个执行的参考画面如图 7.11 所示。

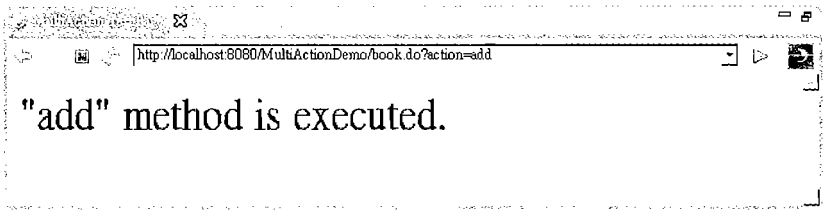


图 7.11　MultiActionDemo 项目的执行结果

7.2.3　MultiActionController 与 PropertiesMethodNameResolver

您也可以将所有方法组织在一个委托（delegate）对象中，而不是编写在 Controller 类中，当请求来到时，委托这个对象执行指定的方法，您只需设置 MultiActionController 的 "delegate" 属性参考至委托对象，例如委托对象可以设计如下：

MultiActionDemo2　　BookDelegate.java

```
package onlyfun.caterpillar;

import javax.servlet.http.*;
import org.springframework.web.servlet.ModelAndView;

public class BookDelegate {
   private String testPage;

   public ModelAndView list(HttpServletRequest req,
                            HttpServletResponse res) {
      return new ModelAndView(
             this.getTestPage(),"executed", "list");
   }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public ModelAndView add(HttpServletRequest req,
                            HttpServletResponse res) {
        return new ModelAndView(
                this.getTestPage(),"executed", "add");
    }

    public ModelAndView delete(HttpServletRequest req,
                               HttpServletResponse res) {
        return new ModelAndView(
                this.getTestPage(),"executed", "delete");
    }

    public String getTestPage() {
        return testPage;
    }

    public void setTestPage(String testPage) {
        this.testPage = testPage;
    }
}
```

现在不用继承 MultiActionController 类来定义 Controller 类，而是在定义文件中直接定义 MultiActionController 的实例：

MultiActionDemo2 mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/book.do">bookController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="paraMethodResolver"
         class="org.springframework.web.servlet.mvc.
                → multiaction.ParameterMethodNameResolver">
        <property name="paramName" value="action"/>
        <property name="defaultMethodName" value="list"/>
    </bean>

   <bean id="bookDelegate"
         class="onlyfun.caterpillar.BookDelegate">
        <property name="testPage" value="test"/>
    </bean>

    <bean id="bookController"
         class="org.springframework.web.servlet.mvc.
                → multiaction.MultiActionController">
        <property name="methodNameResolver"
                ref="paraMethodResolver"/>
        <property name="delegate" ref="bookDelegate"/>
    </bean>

</beans>
```

您可以使用 MultiActionDemo 项目中的 *test.jsp* 网页来呈现结果，这个程序的执行结果与 MultiActionDemo 项目是相同的，可以参考图 7.11 的画面。

上面的程序，也可以使用 PropertiesMethodNameResolver，例如将 Bean 定义文件改写为以下的内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/*book.do">bookController</prop>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="propMethodResolver"
        class="org.springframework.web.servlet.mvc.
                → multiaction.PropertiesMethodNameResolver">
        <property name="mappings">
            <props>
                <prop key="/listbook.do">list</prop>
                <prop key="/addbook.do">add</prop>
                <prop key="/deletebook.do">delete</prop>
            </props>
        </property>
    </bean>

    <bean id="bookDelegate"
         class="onlyfun.caterpillar.BookDelegate">
        <property name="testPage" value="test"/>
    </bean>

    <bean id="bookController"
         class="org.springframework.web.servlet.mvc.
                → multiaction.MultiActionController">
        <property name="methodNameResolver" ref="propMethodResolver"/>
        <property name="delegate" ref="bookDelegate"/>
    </bean>
</beans>
```

根据以上的设置，所有以 book.do 为结尾的请求都会交给 bookController 来处理，而在请求 */listbook.do* 时，会执行 bookDelegate 的 list() 方法，依同样的道理，可以分别使用 */addbook.do* 或 */deletebook.do* 来执行 add() 或 list() 方法，一个执行的参考画面如图 7.12 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

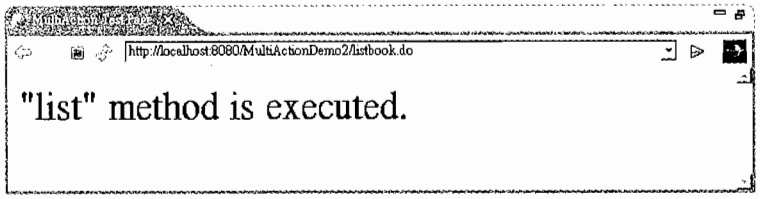


图 7.12　使用 PropertiesMethodNameResolver 的执行结果

7.2.4　ParameterizableViewController

ParameterizableViewController 类是 AbstractController 类的子类，可以直接在 Bean 定义文件中定义并设置它的 "viewName" 属性。当这个 Controller 执行时，会根据 "viewName" 的设置直接导向指定的网页，使用它的好处是可以避免直接对 JSP 网页请求，例如可以设计一个 *index.jsp*，为了让所有的请求都通过 DispatcherServlet 控制，可以将 *index.jsp* 放置在 *WEB-INF* 下，并直接在 Bean 定义文件中这么定义：

```
...
<beans>
   <bean id="urlMapping"
        class="org.springframework.web.servlet.
               → handler.SimpleUrlHandlerMapping">
      <property name="mappings">
         <props>
            <prop key="/index.do">indexController</prop>
           ....
         </props>
      </property>
   </bean>

   <bean id="viewResolver"
        class="org.springframework.web.servlet.
               → view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
   </bean>

   <bean id="indexController"
        class="org.springframework.web.servlet.
               → mvc.ParameterizableViewController">
      <property name="viewName" value="index"/>
   </bean>
<beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

依以上的设置，当连接的网址请求是 */index.do* 时，则会连接至所指定的 */WEB-INF/jsp/index.jsp* 网页。

7.2.5 AbstractCommandController

在使用 Controller 或是 AbstractController 时遇到用户提供请求参数，则必须自行从 HttpServletRequest 对象中取得请求参数，如果打算将这些请求参数设置至某个对象，则必须自行编写相关逻辑或进行类型转换等动作。

可以继承 org.springframework.web.servlet.mvc.AbstractCommandController 来实现 Controller 类，并提供一个 Command 对象，当用户请求时，AbstractCommandController 会试着比对请求参数名称与 Command 上的 Setter 方法名称（也就是 setXXX 方法名称），并调用对应的 Setter 方法设置请求参数值，如果 Setter 上的参数是基本类型，也会自动转换请求参数值为对应的类型。

举个例子来说，假设您设计了一个 User 类作为 Command 类：

AbstractCommandDemo　　User.java

```
package onlyfun.caterpillar;

public class User {
    private String name;
    private String password;

    public void setName(String name) {
        this.name = name;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public String getPassword() {
        return password;
    }
}
```

如果请求中包括了 "name" 与 "password" 参数，而这些参数想要封装为 User 实例，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

则当继承 AbstractCommandController 实现 Controller 时，可以使用 setCommandClass() 来指定 Command 类，并重新定义它的 handle() 方法，例如：

AbstractCommandDemo　　UserController.java

```
package onlyfun.caterpillar;

import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.*;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.
                mvc.AbstractCommandController;
import org.springframework.validation.BindException;

public class UserController extends AbstractCommandController {
    private String userPage;

    public UserController() {
        setCommandClass(User.class);
    }

    protected ModelAndView handle(HttpServletRequest request,
                    HttpServletResponse response,
                    Object command,
                    BindException exception) throws Exception {
        User user = (User) command;

        Map model = new HashMap();
        model.put("name", user.getName());
        model.put("password", user.getPassword());

        return new ModelAndView(getUserPage(), "userInfo", model);
    }

    public void setUserPage(String userPage) {
        this.userPage = userPage;
    }

    public String getUserPage() {
        return userPage;
    }
}
```

在 UserController 的 handle() 方法中，直接取得了 Command 的实例，并在转换操作

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接口为 User 之后，直接操作 getName() 与 getPassword() 方法来取得请求参数。在以上过程中使用了一个 Map 对象来收集信息，以在 JSP 网页中呈现这些信息，假设 JSP 网页设计如下：

AbstractCommandDemo　　userInfo.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>AbstractCommandController Demo</title>
</head>
<body>
    <h1>名称：${userInfo['name']}</h1>
    <h1>密码：${userInfo['password']}</h1>
</body>
</html>
```

接着可以在 Bean 定义文件中设置相关的 Bean 信息，如下所示：

AbstractCommandDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
          class="org.springframework.web.servlet.
                 → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/user.do">userController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.
                 → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    </bean>

    <bean id="userController"
         class="onlyfun.caterpillar.UserController">
        <property name="userPage" value="userInfo"/>
    </bean>

</beans>
```

来看看一个执行的结果画面（如图 7.13 所示），在请求时直接在地址栏附上请求参数：

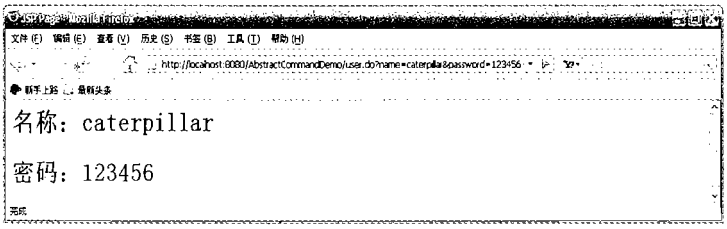


图 7.13　AbstractCommandDemo 项目的执行结果

7.2.6　AbstractFormController

对于简单的表单需求，通常通过继承 org.springframework.web.servlet.mvc.SimpleFormController 来定义 Controller，并重新定义 onSubmit() 或 doSubmitAction() 方法来完成表单功能，而实际上真正的表单处理流程定义在 org.springframework.web.servlet.mvc.AbstractFormController 上，在 API 文件中，有说明其一连串的方法之作用，这里整理一下当中的说明：

1. 执行 formBackingObject() 方法，要回传一个 commandClass 的实例。
2. 执行 initBinder() 方法，允许您对 Command 特定的 Field 作格式化等动作，例如日期格式、金钱格式等，提供非基本类型或 String 的设置。
3. 执行 showForm() 方法，返回一个 View 对象，用于表单的展现。
4. 如果 "bindOnNewForm" 属性被设置为 true，则应用 ServletRequestDataBinder 根据初始请求参数来填入一个新的表单对象，并且执行 onBindOnNewForm() 方法。
5. showForm() 方法会执行 referenceData() 方法，如果在展现表单的过程中，需要一些相关数据，可以回传一个 Map 对象，例如：

```
protected Map referenceData(
        HttpServletRequest request) throws Exception {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    Map ref = new HashMap();
    ref.put("defalutUser", "guest");
    return ref;
}
```

这样一来，就可以在 JSP 网页中使用 ${defaultUser} 的写法来呈现数据，例如所取得的值会是 "guest"。

6. Model 对象被展现，用户可以看到表单并进行填写。

用户填写完表单之后送出表单，将会依以下的流程进行：

1. 接受用户的表单送出，通常使用 POST，如果不想用 POST 这样的方式来判断用户是否送出表单，可以重新定义 isFormSubmission() 方法。
2. 如果 "sessionForm" 属性没有设置，则会执行 formBackingObject() 方法以取得表单对象，否则尝试从 session 中取得表单对象，如果没有找到则执行 handleInvalidSubmit() 方法，默认的动作是尝试建立新的表单对象并重新填写表单。
3. ServletRequestDataBinder 根据目前的请求参数填入表单对象。
4. 执行 onBind() 方法，在绑定数据之后、验证数据之前可以对表单对象进行一些自制的修改动作。
5. 如果 "validateOnBinding" 属性被设置，则使用验证器。
6. 执行 onBindAndValidate() 方法，允许您在验证数据之后对表单对象作一些处理。
7. 表单对象处理完毕，最后执行 processFormSubmission() 方法，子类要重新定义这个方法，以完成用户的请求。

7.2.7 SimpleFormController

org.springframework.web.servlet.mvc.SimpleFormController 是 AbstractFormController 的子类。简单的表单可以直接继承这个类，并重新定义 onSubmit() 或 doSubmitAction() 方法以处理用户的请求，SimpleFormController 的处理流程是这样的：

1. 执行 processFormSubmission() 以检验 Errors 对象，看看在绑定或验证时有无任何的错误。
2. 如果有错误发生，回传 "formView" 所设置的页面。
3. 如果 isFormChangeRequest() 根据 request 被重新定义并回传 true，则也会回到 "formView"，在回到 "formView" 之前会执行 onFormChange()，让您有机会修正表单对象。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

4. 如果没有错误发生，则会执行带有 HttpServletRequest、HttpServletResponse、Object、BindException 参数的 onSubmit()，默认是执行仅带 Object 及 BindException 的 onSubmit()，而后者默认是执行只有 Object 参数的 onSubmit()，默认是导向设置的 "successView"，可以考虑重新定义 doSubmitAction()，不用返回 ModelAndView，默认会导向 "successView"，在不需要向 "successView" 传送任何 Model 数据对象时可以使用。

以下实现一个表单登录程序，以示范 SimpleFormController 的作用，首先设计一个 *form.jsp*：

SimpleFormDemo　　form.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Login Form</title>
</head>
<body>
    <h1>登录表单</h1>
     请输入用户名称与密码: <p>
    <form name="loginform"
         action="/SimpleFormDemo/login.do" method="POST">
    名称 <input type="text" name="username"/><br>
    密码 <input type="password" name="password"/><br>
       <input type="submit" value="确定"/>
    </form>
    注意: 输入错误会再回到这个页面中。

</body>
</html>
```

表单中传递两个参数 "username" 与 "password"，注意将表单的 "method" 属性设置为 "POST"，这样 Spring 才会将请求参数填入表单对象中。先看看 LoginController 的编写：

SimpleFormDemo　　LoginController.java

```
package onlyfun.caterpillar;

import org.springframework.web.servlet.
                    mvc.SimpleFormController;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.web.servlet.*;

public class LoginController extends SimpleFormController {
    public LoginController() {
        setCommandClass(LoginForm.class);
    }

    protected ModelAndView onSubmit(
              Object command) throws Exception {
       LoginForm form = (LoginForm) command;

       if("caterpillar".equals(form.getUsername()) &&
          "123456".equals(form.getPassword())) {
          return new ModelAndView(
                getSuccessView(),"user", form.getUsername());
       }
       else {
           return new ModelAndView(getFormView());
       }
    }
}
```

onSumit() 方法接收 command 对象，这个对象代表了表单对象，getSuccessView() 与 getFormView() 是 SimpleFormController 中定义的两个方法，分别表示通过验证与失败的目标网页，待会可以直接在 Bean 定义文件中定义，先来看看表单对象，它只是个简单的 JavaBean：

SimpleFormDemo　　LoginForm.java

```
package onlyfun.caterpillar;

public class LoginForm {
    private String username;
    private String password;

    public void setUsername(String username) {
       this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUsername() {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return username;
    }

    public String getPassword() {
        return password;
    }
}
```

接下来看看 Bean 定义文件是如何编写的：

SimpleFormDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
          class="org.springframework.web.servlet.
                 → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/login.do">loginController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.
                 → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="loginController"
          class="onlyfun.caterpillar.LoginController">
        <property name="successView" value="success"/>
        <property name="formView" value="form"/>
    </bean>

</beans>
```

定义文件中定义了 "successView" 与 "formView"，其中 *form.jsp* 之前已经编写完成，来看一下 *success.jsp* 的编写：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

SimpleFormDemo　　success.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Login Success</title>
</head>
<body>
    <H1>哈啰！ ${user}！！</H1>
     这是您的神秘礼物！^o^
</body>
</html>
```

直接请求 login.do 的话，由于未通过登录检查，所以会回传表单登录画面，如图 7.14 所示。

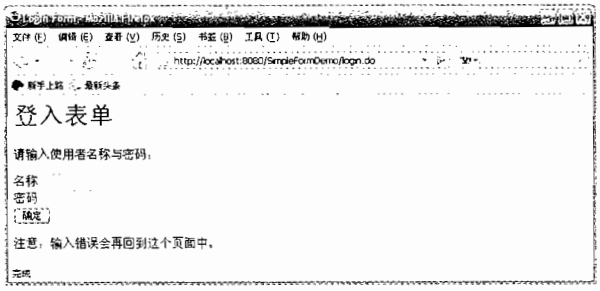


图 7.14　SimpleFormDemo 项目的登录表单

登录成功，则会进入 *success.jsp* 并显示用户的名称，登录失败的话，依 "formView" 的设置，会回到原来的登录表单，登录成功的画面如图 7.15 所示。

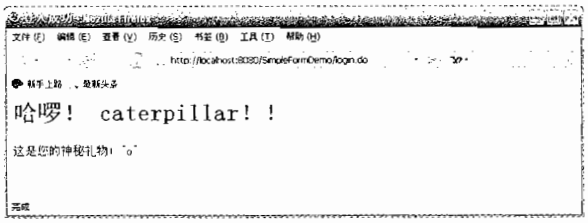


图 7.15　SimpleFormDemo 项目的登录成功画面

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.2.8　AbstractWizardFormController

org.springframework.web.servlet.mvc.AbstractWizardFormController 可以让您设计出如桌面应用程序的向导（Wizard），通常用于问卷表单内容相当长时，与其使用一个表单呈现所有的问卷内容，不如将问卷内容分作数个画面，让用户一页一页完成问卷，您可以继承 AbstractWizardFormController，并重新定义 processFinish() 方法，它定义的是全部问卷提交后的处理，例如：

AbstractWizardDemo　　WizardController.java

```
package onlyfun.caterpillar;

import java.util.*;
import javax.servlet.http.*;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.
               mvc.AbstractWizardFormController;
import org.springframework.validation.BindException;

public class WizardController
             extends AbstractWizardFormController {
   private String successView;

   public WizardController() {
      setCommandClass(Questionnaire.class);
   }

   protected ModelAndView processFinish(
                     HttpServletRequest request,
                     HttpServletResponse response,
                     Object command,
                    BindException exception) throws Exception {
      Questionnaire questionnaire = (Questionnaire) command;

      Map model = new HashMap();
      model.put("q1", questionnaire.getQuestion1());
      model.put("q2", questionnaire.getQuestion2());
      model.put("q3", questionnaire.getQuestion3());

      return new ModelAndView(getSuccessView(), "ans", model);
   }

   public void setSuccessView(String successView) {
      this.successView = successView;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    public String getSuccessView() {
        return successView;
    }
}
```

所有的问卷结果收集在一个 Command 对象中，上面的程序只是取得结果并将它设置在 Model 对象中，以在 View 上呈现数据，Command 的类设计如下：

AbstractWizardDemo　　Questionnaire.java

```
package onlyfun.caterpillar;

public class Questionnaire {
    private String question1;
    private String question2;
    private String question3;

    public void setQuestion1(String question1) {
        this.question1 = question1;
    }

    public void setQuestion2(String question2) {
        this.question2 = question2;
    }

    public void setQuestion3(String question3) {
        this.question3 = question3;
    }

    public String getQuestion1() {
        return question1;
    }

    public String getQuestion2() {
        return question2;
    }

    public String getQuestion3() {
        return question3;
    }
}
```

问卷的流程规范在 Bean 定义文件中，通过设置 AbstractWizardFormController 的

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

"pages" 属性来决定，例如：

AbstractWizardDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                ➞ handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/wizard.do">wizardController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                ➞ view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

   <bean id="wizardController"
         class="onlyfun.caterpillar.WizardController">
        <property name="successView" value="success"/>
        <property name="pages">
            <list>
                <value>start</value>
                <value>question1</value>
                <value>question2</value>
                <value>question3</value>
            </list>
        </property>
    </bean>

</beans>
```

根据以上的设置，问卷将按照 *start.jsp*、*question1.jsp*、*question2.jsp*、*question3.jsp* 的顺序来完成，首先看看 *start.jsp* 的编写：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

AbstractWizardDemo start.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Insert title here</title>
</head>
<body>
    <form name="questionform"
          action="/AbstractWizardDemo/wizard.do" method="POST">
          欢迎您填写问卷!
        <input type="submit" value="Start" name="_target1"/>
    </form>
</body>
</html>
```

当您连接至 Wizard 控制器时，默认会取得设置文件中索引 0 位置的网页，第一个页面通常是欢迎或问卷说明页面，决定下一个要展示的页面之方法在于请求中有无一个以 "_target" 开头，并跟着一个号码的请求参数，例如在 "_target1" 这样的设置中，号码表示在设置文件中 <list> 的顺序，上面的网页在点击 Start 按钮后，会显示第二个页面：

AbstractWizardDemo question1.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Question 1</title>
</head>
<body>
    <form name="questionform1"
          action="/AbstractWizardDemo/wizard.do" method="POST">
        问题一 <input type="text" name="question1"/><br>
        <input type="submit" value="Previous" name="_target0"/>
        <input type="submit" value="Next" name="_target2"/>
    </form>
</body>
</html>
```

由 "_target" 与号码来决定显示的页面是 getTaretPage() 方法所默认的，必要的话，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

也可以重新定义这个方法以决定下一个页面由哪一个参数决定，在 *question1.jsp* 的设计中，点击 Previous 按钮的话，会返回问卷的上一页，点击 Next 的按钮，会返回问卷的下一页，*question2.jsp* 的设计也类似，为节省篇幅，请自行观看光盘中 AbstractWizardDemo 项目的 *question2.jsp* 文件。

决定是否完成问卷的方式，在于检验请求中是否有请求参数 "_finish"，如果检查到这个值，就会执行 processFinish() 方法，*question3.jsp* 是最后一个问卷，来看看它是如何编写的：

AbstractWizardDemo　　question3.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Question 3</title>
</head>
<body>
    <form name="questionform3"
          action="/AbstractWizardDemo/wizard.do" method="POST">
        問題三 <input type="text" name="question3"/><br>
        <input type="submit" value="Previous" name="_target2"/>
        <input type="submit" value="Finish" name="_finish"/>
    </form>
</body>
</html>
```

记得最后一个送出的问卷要带有一个 "_finish" 参数，它才会执行 AbstractWizardFormControlle 的 processFinish() 方法，最后设计一个 *success.jsp* 来显示结果：

AbstractWizardDemo　　success.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Success</title>
</head>
<body>
    <H1>Q1: ${ans.q1}</H1> <br>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <H1>Q2: ${ans.q2}</H1> <br>
    <H1>Q3: ${ans.q3}</H1> <br>
</body>
</html>
```

一个问卷提交后的结果画面如图 7.16 所示。

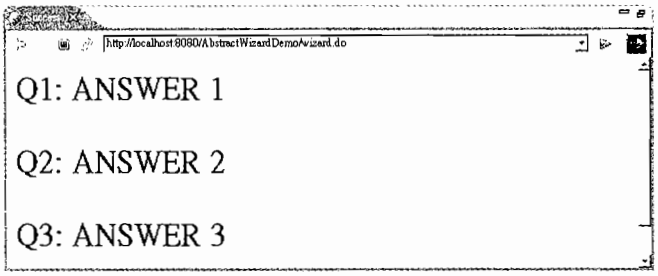


图 7.16 AbstractWizardDemo 项目的提交结果画面

您也可以在问卷中设计一个 Cancel 按钮，在点击该按钮后，送出 "_cancel" 请求参数，例如：

```
....
    <form name="questionform3"
         action="/AbstractWizard/wizard.do" method="POST">
        ....
        <input type="submit" value="Cancel" name="_cancel"/>
    </form>
....
```

当请求中带有 "_cancel" 请求参数时，则会执行 processCancel() 方法，您可以重新定义这个方法，例如将用户送回问卷开始时的页面：

```
...
    protected ModelAndView processCancel(
                HttpServletRequest request,
                HttpServletResponse response,
                Object command,
                BindException exception) throws Exception {
        return new ModelAndView("start");
    }
...
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.2.9　ThrowawayController

org.springframework.web.servlet.mvc.throwaway.ThrowawayController 并不在 Spring 的 Controller 继承架构中，它是一个独立的接口定义，实现该接口的类不是以单例的方式产生并重复使用的，ThrowawayController 在每一次请求时都要产生新的对象，且用了就丢，不会重复使用，这也是名称上带有"Throwaway"的原因（ThrowawayController 提供的是类似 Webwork 形式的 Action 对象）。

ThrowawayController 接口的定义如下：

```
package org.springframework.web.servlet.mvc.throwaway;

import org.springframework.web.servlet.ModelAndView;

public interface ThrowawayController {
    ModelAndView execute() throws Exception;
}
```

实现 ThrowawayController 接口的类必须实现 execute() 方法，可以看到 execute() 方法中不带任何的参数，这使得 ThrowawayController 易于测试，不需要像 Controller 必须有 HttpServletRequest、HttpServletResponse 的实例。

execute() 方法上没有任何的参数，如果要收集请求参数，ThrowawayController 本身也兼具 Command 的作用（这点熟悉 Webwork 的人并不陌生），例如可以如下实现一个 ThrowawayController：

```
....
public class SomeController implements ThrowawayController {
    private String someParam;

    public ModelAndView execute() throws Exception {
        ....
        return new ModelAndView(...);
    }

    public void setSomeParam(String someParam) {
        this.someParam = someParam;
    }

    ....
}
```

当请求参数中带有 "someParam" 的参数时，会将之设置至 ThrowawayController 对应

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

的 setSomeParam() 方法中。

在设置 Bean 定义文件时必须注意的是，ThrowawayController 必须将 "scope" 属性设置为 "singleton"，每一次的请求要产生一个新的实例，用过即丢，例如：

```
...
   <bean id="someController"
        class="onlyfun.caterpillar.SomeController"
        scope="singleton">
       ....
   </bean>
...
```

DispatcherServlet 默认会使用 org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter，这使得 Controller 继承体系下的 Controller 类都可以直接为 DispatcherServlet 使用，但为了要告诉 DispatchServlet 使用 ThrowawayController，必须在 Bean 定义文件中声明，例如：

```
...
<bean id="throwawayHandler"
     class="org.springframework.web.servlet.mvc.
          → throwaway.ThrowawayControllerHandlerAdapter"/>

   <bean id="simpleHandler"
        class="org.springframework.web.servlet.
          → mvc.SimpleControllerHandlerAdapter"/>
...
```

在上面的设置中，DispatcherServlet 可以使用 ThrowawayController，如果您要混合 Controller 接口的继承体系，则再加入 SimpleControllerHandlerAdapter，如此两种 Handler 就可以混合使用了。

7.3 搭配 Controller 的相关类

在这节中，将介绍如何在 Controller 上搭配使用验证器（Validator）、如何实现 Command 数据的类型转换，以及如何使用 Spring 的 API 来实现文件上传功能。

7.3.1 实现 Validator

使用 SimpleFormController 时，可以搭配一个 org.springframework.validation.Validator 接口的实现对象，帮助您作基本的服务器端数据验证工作，可以设计一个通用的 Validator，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

将之作为 Bean 注入给需要进行表单验证的其他 Bean 对象。

Validator 接口有两个必须实现的方法，其定义如下所示：

```
package org.springframework.validation;

public interface Validator {
    boolean supports(Class clazz);
    void validate(Object obj, Errors errors);
}
```

supports() 方法回传一个 boolean 值，表示是否支持对所传入的对象进行验证，只有在传回 true 的情况下，才会使用 validate() 方法进行验证工作，在 validate() 方法的参数中，obj 表示传入的表单对象，您可以对它进行一些验证，如果有错误的话，可以使用 Errors 的 reject() 或 reject-Value() 等方法加入错误消息。在后续的处理中，若 errors 对象中包括错误消息，会回到 getViewPage() 所设置的页面。

来看一个简单的实现：

```
package onlyfun.caterpillar;

import org.springframework.validation.Validator;
import org.springframework.validation.Errors;

public class LoginValidator implements Validator {
    public boolean supports(Class clazz) {
        return clazz.equals(LoginForm.class);
    }

    public void validate(Object obj, Errors errors) {
        LoginForm form = (LoginForm) obj;

        if(form.getPassword().length() < 4 ) {
            errors.rejectValue("password",
                "less-than-four", null, "密码不得小于四个字元");
        }
    }
}
```

这个 Validator 类可以搭配 7.2.7 节介绍的 SimpleFormDemo 项目来使用，验证通常是在检查数据的完备性、安全性等条件，诸如检查密码的格式就是一例，至于用户名称与密码是否正确，这应该是 Controller 的处理工作。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

搭配 SimpleFormController 来使用 Validator，只要在 Bean 定义文件中加入 Validator 的 Bean 定义，以及让 SimpleFormController 参考到它即可，例如：

```
...
<bean id="loginValidator"
      class="onlyfun.caterpillar.LoginValidator"/>

<bean id="loginAction"
      class="onlyfun.caterpillar.LoginAction">
   <property name="commandClass"
      value="onlyfun.caterpillar.LoginForm"/>
   <property name="validator" ref="loginValidator"/>
   <property name="successView" value="success"/>
   <property name="formView" value="form"/>
</bean>
...
```

"validator" 属性参考至 loginValidator 实例，只要有设置 "validator"，SimpleFormController 就会使用它。

在进行验证时，您可以使用 org.springframework.validation.ValidationUtils，上面有一些方便的静态方法，像是 rejectIfEmpty()、rejectIfEmptyOrWhitespace() 等，可以查查联机 API 文件看如何使用。

另一方面，对于 AbstractWizardFormController，可以重新定义 validatePage() 方法来进行验证，例如：

```
...
   protected void validatePage(Object command,
                               Errors errors, int page) {
      (YourCommand) your = (YourCommand) command;
      switch(page) {
          case 1:
              validator.validateSome(your, errors);
              break;
          case 2:
              validator.validateOther(your, errors);
              break;
           ....
      }
   }
...
```

每进行一页 Wizard 表单，都会调用该方法一次，让您针对该次送出的数据进行验证，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

"page" 参数表示该次送出表单的页数，与 SimpleFormController 的 Validator 不同的是，AbstractWizardFormController 不会自动执行 Validator 的 validate() 方法，您要根据当次页面的数据进行对应的方法验证，通常采取如上的委托对象，将验证交由一个 Validator 对象来执行。

7.3.2　使用 PropertyEditor

对于 BaseCommandController 及其子类来说，它的 Command 对象并不一定要接受基本类型或是 String 类型，您可以编写一个实现 java.beans.PropertyEditor 的类，并在当中进行转换，例如将接收到的字符串转换为 User 类的实例。

举个实例来说，假设 Command 对象如下设计：

PropertyEditorDemo　　SomeForm.java

```
package onlyfun.caterpillar;

public class SomeForm {
    private User user;

    public void setUser(User user) {
        this.user = user;
    }

    public User getUser() {
        return user;
    }
}
```

SomeForm 为网页表单的对象代表，而其中出现的 User 类如下设计：

PropertyEditorDemo　　User.java

```
package onlyfun.caterpillar;

public class User {
    private String firstName;
    private String lastName;

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

Command对象将接收一个自制类型的User实例，然而从HTTP接收到的参数值是String类型，您可以编写一个UserPropertyEditor类做转换，通常直接继承java.beans.PropertyEditorSupport并重新定义getAsText()及setAsText()方法，例如：

PropertyEditorDemo　　UserPropertyEditor.java

```
package onlyfun.caterpillar;

import java.beans.PropertyEditorSupport;

public class UserPropertyEditor extends PropertyEditorSupport {
    public String getAsText() {
        Object o = this.getValue();

        if(o == null || !(o instanceof User)) {
            return null;
        }

        User user = (User) o;
        String name = user.getFirstName()
                    + "," + user.getLastName();
        return name;
    }

    public void setAsText(String text) {
        String[] tokens = text.split(",");

        User user = new User();
        user.setFirstName(tokens[0]);
        user.setLastName(tokens[1]);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            setValue(user);
        }
    }
```

当必须将指定的对象转换为字符串时，会执行 getAsText() 方法，而接收到的参数要将之转换为指定的对象时，会执行 setAsText() 方法，接着来编写一个测试的 Controller：

PropertyEditorDemo　　SomeFormController.java

```
package onlyfun.caterpillar;

import org.springframework.web.servlet.
                    mvc.SimpleFormController;
import org.springframework.web.servlet.*;
import java.util.*;
import javax.servlet.http.HttpServletRequest;
import org.springframework.web.
                    bind.ServletRequestDataBinder;

public class SomeFormController extends SimpleFormController {
    public SomeFormController() {
        setCommandClass(SomeForm.class);
    }

    protected ModelAndView onSubmit(
                    Object command) throws Exception {
        SomeForm form = (SomeForm) command;

        Map model = new HashMap();
        model.put("firstName",
                form.getUser().getFirstName());
        model.put("lastName",
                form.getUser().getLastName());

        return new ModelAndView(this.getSuccessView(), model);
    }

    protected void initBinder(HttpServletRequest request,
                        ServletRequestDataBinder binder)
                                    throws Exception {
        super.initBinder(request, binder);
        binder.registerCustomEditor(
                User.class, new UserPropertyEditor());
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

注意到这里重新定义了 initBinder() 方法，并在当中使用 ServletRequest-DataBinder 的 registerCustomEditor() 方法注册自制的 PropertyEditor，来看看定义文件的内容：

PropertyEditorDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
          class="org.springframework.web.servlet.
               → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/index.do">indexController</prop>
                <prop key="/someForm.do">someFormController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
          class="org.springframework.web.servlet.
               → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="indexController"
          class="org.springframework.web.servlet.
                    → mvc.ParameterizableViewController">
        <property name="viewName" value="index"/>
    </bean>

    <bean id="someFormController"
          class="onlyfun.caterpillar.SomeFormController">
        <property name="successView" value="hello"/>
    </bean>

</beans>
```

现在假设编写有一个测试网页：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

PropertyEditorDemo　　index.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Form</title>
</head>
<body>
    <form name="someForm"
          action="/PropertyEditorDemo/someForm.do" method="POST">
        用户 <input type="text" name="user"/><br>
        <input type="submit" value="Submit"/>
    </form>
</body>
</html>
```

在输入字段中设置了"user"参数，当数据送出后，会经由 PropertyEditor 的转换，假设呈现处理结果的 *hello.jsp* 网页如下编写：

PropertyEditorDemo　　hello.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>User Info</title>
</head>
<body>
    <h1>${firstName} - ${lastName}</h1>
</body>
</html>
```

如果在 *index.jsp* 填入 "Justin,Lin"，则结果会显示以下的内容，如图 7.17 所示。

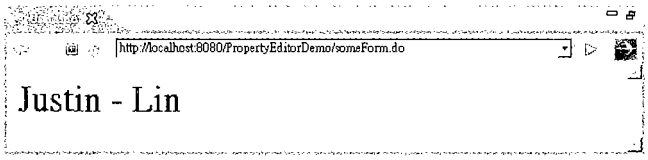


图 7.17　PropertyEditorDemo 项目的结果画面

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

7.3.3 文件上传

如果想要进行文件上传的动作，则可以使用实现 org.springframework.web.multipart.MultipartResolver 接口的类，Spring 提供 org.springframework.web.multipart.commons.CommonsMultipartResolver 与 org.springframework.web.multipart.cos.CosMultipartResolver，它们分别支持 Commands FileUpload 及 COS FileUpload。

举个实际的例子，假设您设计了一个 FileForm 类：

FileUploadDemo　　FileForm.java

```
package onlyfun.caterpillar;

public class FileForm {
    private String name;
    private byte[] contents;

    public void setName(String name) {
        this.name = name;
    }

    public void setContents(byte[] contents) {
        this.contents = contents;
    }

    public String getName() {
        return name;
    }

    public byte[] getContents() {
        return contents;
    }
}
```

FileForm 中接受文件上传的属性类型是 byte 数组，用户上传的文件可以通过 org.springframework.web.multipart.support.ByteArrayMultipartFileEditor 转换为 byte 数组并设置给 FileForm，这必须在定义 Controller 时注册，例如：

FileUploadDemo　　UploadController.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.bind.ServletRequestDataBinder;
import org.springframework.web.multipart.
                    support.ByteArrayMultipartFileEditor;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.
                              mvc.SimpleFormController;

public class UploadController extends SimpleFormController {
   private String path;

   public UploadController() {
      setCommandClass(FileForm.class);
   }

   protected ModelAndView onSubmit(
                       HttpServletRequest request,
                       HttpServletResponse response,
                       Object command,
                       BindException errors)
                                 throws Exception {
      FileForm form = (FileForm) command;

      String storedPath = path +
                       System.getProperty("file.separator") +
                       form.getName();

      BufferedOutputStream bufferedOutputStream =
            new BufferedOutputStream(
                  new FileOutputStream(storedPath));

      bufferedOutputStream.write(form.getContents());
      bufferedOutputStream.close();

      return new ModelAndView(getSuccessView(),
                          "filename", form.getName());
   }

   protected void initBinder(HttpServletRequest request,
                     ServletRequestDataBinder binder)
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                                        throws Exception {
        super.initBinder(request, binder);
        binder.registerCustomEditor(
                byte[].class, new ByteArrayMultipartFileEditor());
    }

    public void setPath(String path) {
        this.path = path;
    }
}
```

依以上的设置，文件上传后会储存在指定的目录下，要使用文件上传的功能，必须在定义文件中加入 MultipartResolver 的定义，可以选择使用 CommonsMultipartResolver 或 CosMultipartResolver，例如以下使用 CommonsMultipartResolver 来定义：

FileUploadDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/index.do">indexController</prop>
                <prop key="/upload.do">uploadController</prop>
            </props>
        </property>
    </bean>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="indexController"
          class="org.springframework.web.servlet.
                → mvc.ParameterizableViewController">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <property name="viewName" value="index"/>
    </bean>

    <bean id="multipartResolver"
         class="org.springframework.web.multipart.
                → commons.CommonsMultipartResolver">
        <property name="maxUploadSize" value="1000000"/>
    </bean>

    <bean id="uploadController"
         class="onlyfun.caterpillar.UploadController">
        <property name="formView" value="form"/>
        <property name="successView" value="success"/>
        <property name="path" value="c:\\upload"/>
    </bean>

</beans>
```

在定义文件中，通过设置 UploadController 的 "path" 属性为 "C:\upload"，表示上传后的文件会储存在 *C:\upload* 目录之中，接着设计一个 *index.jsp*：

FileUploadDemo　　index.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Upload Page</title>
</head>
<body>
    <form name="uploadForm" enctype="multipart/form-data"
         action="/FileUploadDemo/upload.do" method="POST">
        上载后文件名称: <input name="name" type="text"/><br>
        选择文件: <input name="contents" type="file"/><br>
        <input type="submit" value="Submit"/>
    </form>
</body>
</html>
```

在这个页面中可以设置上传后的文件名称，并可以选择所要上传的文件，*success.jsp* 则设计如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

FileUploadDemo success.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Upload Success</title>
</head>
<body>
    <h1>File: ${filename} upload successfully.</h1>
</body>
</html>
```

良葛格的话匣子<<<

这个项目使用到相依赖的 Commands FileUpload，所以要在 *lib* 目录中包括 *commons-fileupload.jar* 以及 *commons-io.jar* 文件。

7.4 接下来的主题

这一个章节对 Spring Web MVC 的主要架构与各种的 `Controller` 及相应 API 作了说明，Spring 在 Web 层所提供的解决方案还不只这些，下一个章节将来看一下 Spring 如何在 View 层技术上提供多种不同的解决方案，以及如何在 Spring 中整合其他的 Web 框架，如 Struts、JSF 等。



图 7.18 Spring 让 MVC 架构的实现更为简单

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第

View 层方案、Web 框架整合

章

当使用 JSP 作为 View 层技术时，可以结合 JSTL 以及 Spring 提供的标签，除了将 JSP 技术作为 View 层，Spring 还提供了不同 View 层技术的方案，您甚至可以定义自己的 View 层技术实现。

在不使用 Spring 提供的 Web MVC 框架时，也可以结合其他的 Web 框架，像是 Struts、JSF 等，在本章中，将来看看在结合其他的 Web 框架时，Spring 所居的角色，以及实际的整合方式。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.1　结合 JSTL 与 Spring 标签

当使用 JSP 作为 View 层技术时，可以结合 JSTL（JavaServer Pages Standard Tag Library）以及 Spring 提供的标签，Spring 所提供的标签可与 Spring MVC Web 框架结合使用，至于页面的呈现逻辑则由 JSTL 来提供。

8.1.1　结合 JSTL

在处理完用户的请求之后，呈现结果页面时必须根据处理的结果来显示不同的画面，直接在 JSP 上使用 scriptlet 编写 Java 程序代码并不是好方法，这会使得页面上标签与程序代码混杂在一起，当所使用的 View 层技术为 JSP 时，可以运用 JSTL 来协助页面逻辑的生成，而不是在 JSP 页面中直接编写 Java 程序代码。

Java EE 之中包括 JSTL，可在 Java 官方网站取得 JSTL 实现：*http://java.sun.com/products/jsp/jstl/*（如图 8.1 所示）。

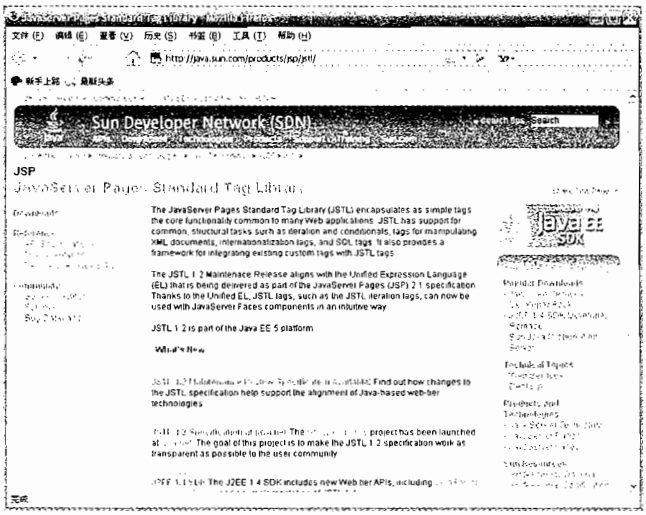


图 8.1　Java 官方网站提供 JSTL 实现

在“Downloads”中提供几个取得 JSTL 的方式，您可以直接下载 Java EE，其中就包括 JSTL 实现，或者仅下载 Jakarta Taglibs Standard tag library，将当中的 *jstl.jar* 与 *standard.jar* 加入至 *lib* 中。如果您先前下载的 Spring 是相依赖版本，也可以在 *lib/j2ee* 目录下找到 *jstl.jar*，在 *lib/jakarta-taglibs* 目录中则可以找到 *standard.jar*。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

以下示范一个完整的范例，结合 Spring 与 JSTL 的功能来进行有条件的页面呈现，首先编写 HelloController 类：

JSTLSpringDemo HelloController.java

```
package onlyfun.caterpillar;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

public class HelloController implements Controller {
    private String viewPage;

    public ModelAndView handleRequest(HttpServletRequest req,
                                      HttpServletResponse res)
                                        throws Exception {
        String user = req.getParameter("user");
        Integer age = new Integer(
                Integer.parseInt(req.getParameter("age")));

        java.util.Map model = new java.util.HashMap();
        model.put("name", user);
        model.put("age", age);

        return new ModelAndView(viewPage, model);
    }

    public void setViewPage(String viewPage) {
        this.viewPage = viewPage;
    }
}
```

在这个程序中，只是简单地收集请求参数，并设置至 Model 对象中，接着在定义文件中，InternalResourceViewResolver 的 "viewClass" 属性可以设置为 org.springframework.web.servlet.view.JstlView，例如：

JSTLSpringDemo mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
        <property name="viewClass"
            value="org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean name="/hello.do"
         class="onlyfun.caterpillar.HelloController">
        <property name="viewPage" value="hello"/>
    </bean>

</beans>
```

来写一个简单的测试页面，在这个当中使用 JSTL 作页面逻辑的呈现判断，设置页面只有在 "age" 大于等于 18 时，才会显示欢迎语句：

JSTLSpringDemo　　hello.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>JSTL And Spring</title>
</head>
<body>
    <c:choose>
        <c:when test="${age >= 18}">
            <h1>嗨, <c:out value="${name}"/> ! </h1>
        </c:when>
        <c:otherwise>
            <h1>未滿十八岁喔! </h1>
        </c:otherwise>
    </c:choose>
</body>
</html>
```

直接在地址栏上给定请求参数，并来看看一个执行时的结果画面（如图 8.2 所示）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

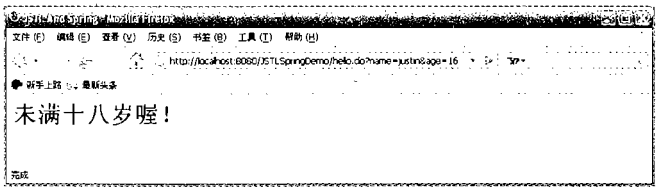


图 8.2 JSTLSpringDemo 项目的执行结果

8.1.2 <spring:bind> 标签

Spring 提供自制的标签库，可以与相关的组件结合，提供页面与表单对象、错误消息的数据绑定功能，为节省篇幅，这里以 7.2.7 节中实现过的 SimpleFormDemo 项目来进行修改扩充，加入 Spring 的 <bind> 标签功能，首先修改一下 *form.jsp*：

SpringBindDemo form.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="spring"
        uri="http://www.springframework.org/tags"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Login Form</title>
</head>
<body>
    <h1>登录表单</h1>
    <spring:bind path="command.*">
        <font color="red">
            <b>${status.errorMessage}</b>
        </font><br>
    </spring:bind>

    请输入用户名称与密码: <p>
    <form name="loginform"
         action="/SpringBindDemo/login.do"
         method="post">

        <spring:bind path="command.username">
        名称 <input type="text"
                  name="${status.expression}"
                  value="${status.value}"/><br>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        </spring:bind>

        <spring:bind path="command.password">
        密码 <input type="password"
                   name="${status.expression}"
                   value="${status.value}"/><br>
        </spring:bind>

        <input type="submit" value="确定"/>
    </form>
    注意：输入错误会再回到这个页面中。
</body>
</html>
```

<spring:bind> 的 "path" 属性设置要绑定的表单对象名称，这个名称是设置在 "loginController" 中的 "commandName" 属性，默认名称是 "command"，当设置为 "command.*" 时，表示绑定表单对象上所有相关的数据，"status" 的 "errorMessage" 会显示 Controller 中设置的错误消息，这待会在 Controller 的实现中会再说明。

在表单中，对于 "username" 字段，绑定了 "command.username" 属性，"status" 的 "expression" 会显示绑定的属性名称，而 "value" 则显示表单对象中所储存的值，这里设计的程序在登录失败后会回到 *form.jsp*，这样可以在同一个页面上显示错误消息与之前输入的值。

为了配合 *form.jsp* 上的绑定标签，可以对 SimpleFormDemo 项目中 LoginController 类作一些修正：

SpringBindDemo　　LoginController.java

```
package onlyfun.caterpillar;

import org.springframework.validation.BindException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.web.servlet.*;

public class LoginController extends SimpleFormController {
    public LoginController() {
        setCommandClass(LoginForm.class);
    }

    protected ModelAndView onSubmit(Object command,
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                                BindException errors)
                                    throws Exception {
        LoginForm form = (LoginForm) command;

        if("caterpillar".equals(form.getUsername()) &&
             "123456".equals(form.getPassword())) {
           return new ModelAndView(getSuccessView(),"user",
                                form.getUsername());
        }
        else {
             // 返回 Form 页面时，要有一个 errors 的 Model 实例
             errors.reject("loginfail", "用户名称或密码错误");
           return new ModelAndView(getFormView(),
                                errors.getModel());
        }
    }
}
```

Spring 标签绑定需要一个 BindException 对象，所以这次使用 onSumit() 的另一个版本，当验证失败时，在 BindException 中使用 reject() 方法，表示拒绝这次输入的数据。reject() 接受两个参数，第一个是 error code，如果有设置 MessageSource（参考 3.4.2 节），这会到设置的 *.properties* 文件中依 error code 为"键（Key）"，以找出相对应的消息，如果没有设置 MessageSource，则使用第二个参数所设置的值作为错误消息输出。

errors 的 getModel() 方法回传一个 Map 对象，之前使用 reject() 所储存的错误消息就包括在这个对象中，它将之设置给 ModelAndView，之后会处理为绑定消息并在标签上输出。

剩下没在书中呈现的 JSP 网页、类与定义文件等，都与 SimpleFormDemo 项目相同，请自行参考 SpringBindDemo 项目中的相关文件内容，来看看还没有登录前 *form.jsp* 的显示内容，如图 8.3 所示。

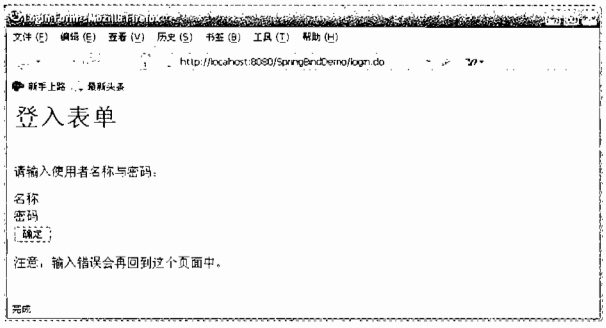


图 8.3 SpringBindDemo 项目的执行结果之一

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

如果登录错误，则会显示以下的内容，绑定标签将错误消息与之前输入的数据显示在对应的标签上，如图 8.4 所示。

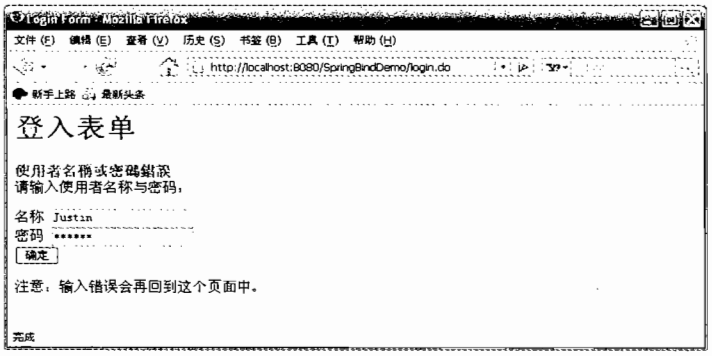


图 8.4　SpringBindDemo 项目的执行结果之二

8.1.3　数据绑定的几个方法

来看看几个数据绑定的方法。您可以一次绑定表单对象的一个属性与数值，例如：

```
...
<spring:bind path="command.username">
名称 <input type="text"
           name="${status.expression}"
           value="${status.value}"/><br>
</spring:bind>

<spring:bind path="command.password">
密码 <input type="password"
           name="${status.expression}"
           value="${status.value}"/><br>
</spring:bind>
...
```

"expression" 会显示被绑定的属性名称，而 value 则显示被绑定的属性值。

或者是这么绑定 Command：

```
...
<spring:bind path="command">
名称 <input type="text"
           name="username"
           value="${command.username}"/><br>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
密码 <input type="password"
           name="password"
           value="${command.password}"/><br>
</spring:bind>
...
```

对于错误消息，之前用 BindException 中的 reject() 方法，但这个方法并不会区分特定字段错误消息，可以使用 rejectValue() 方法来加入错误消息：

```
rejectValue(String field, String errorCode,
            Object[] errorArgs, String defaultMessage)
rejectValue(String field, String errorCode,
            String defaultMessage)
```

rejectValue() 的 field 参数让您指定表单对象的属性，errorCode 参数指定资源文件中的“键”，errorArgs 参数用于指定资源文件中的占位字符，而默认消息则找不到资源文件时所要呈现的消息。一个使用例子如下：

```
....
public void ModelAndView(....,
                BindException errors) throws Exception {
    ...
    errors.rejectValue("username",
                "error", null, "用户名称错误");
    ...
    errors.rejectValue("password", "error", null, "密码错误");

    return new ModelAndView(
                this.getFormView(), errors.getModel());
}
...
```

如上面的程序片段中加入了消息，可以使用绑定标签来呈现消息：

```
...
<spring:bind path="command.username">
名称 <input type="text"
           name="${status.expression}"
           value="${status.value}"/><br>
<font color="red">${status.errorMessage}</color>
</spring:bind>

<spring:bind path="command.password">
密码 <input type="password"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            name="${status.expression}"
            value="${status.value}"/><br>
<font color="red">${status.errorMessage}</color>
</spring:bind>
...
```

相应的错误消息会绑定至相关的字段，如果同一个属性上被绑定了错误消息，则可以使用 ${status.errorMessages} 取出，搭配 JSTL 来输出消息，例如：

```
...
<spring:bind path="command.username">
名称 <input type="text"
            name="${status.expression}"
            value="${status.value}"/><br>
<c:if test="${status.error}">
    <font color="red">
    错误: <br>
        <c:forEach items="${status.errorMessages}"
                   var="error">
            <c:out value="${error}"/><br>
        </c:forEach>
    </font>
</c:if>
</spring:bind>
...
```

${status.error} 可用于测试是否有错误消息，也可以使用 <spring: hasBind-Errors> 来决定是否输出某些内容，例如：

```
...
<spring:hasBindErrors>
    发现以下的错误，请更正...
    ......
</spring:hasBindErrors>
...
```

或者也可以不区分表单对象属性，一次输出所有的错误消息，例如：

```
....
<spring:bind path="command.*">
    <font color="red">
    错误: <br>
        <c:forEach items="${status.errorMessages}"
                   var="error">
            <c:out value="${error}"/><br>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        </c:forEach>
    </font>
</spring:bind>
...
```

8.1.4 <spring:message> 标签

可以使用 <spring:message> 标签并结合 ResourceBundleMessageSource 在网页上显示 *messages.properties* 中的文字消息，例如在 *messages.properties* 中定义如下：

MessageTagDemo　　messages.properties

```
welcome=Hello, {0} {1}. Welcome to Spring.
```

须记得 *messages.properties* 文件必须在 *Classpath* 的路径下，例如在 */WEB-INF/classes/* 下，Bean 定义文件中要加入 ResourceBundleMessageSource 的定义，如下所示：

MessageTagDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
       <property name="mappings">
          <props>
             <prop key="/index.do">indexController</prop>
          </props>
       </property>
    </bean>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
       <property name="prefix" value="/WEB-INF/jsp/"/>
       <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="messageSource"
         class="org.springframework.context.
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                → support.ResourceBundleMessageSource">
        <property name="basename" value="messages"/>
    </bean>

    <bean id="indexController"
              class="org.springframework.web.servlet.
                → mvc.ParameterizableViewController">
        <property name="viewName" value="index"/>
    </bean>

</beans>
```

在这里直接使用了 ParameterizableViewController 来转发用户的请求至 *index.jsp* 网页，网页的内容如下所示：

MessageTagDemo　　index.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="spring"
        uri="http://www.springframework.org/tags"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Message Tag Demo</title>
</head>
<body>
  <h1><spring:message code="welcome" arguments="Justin,Lin"/></h1>
</body>
</html>
```

"arguments" 中指定的参数，以逗号分隔，会分别取代 *messages.properties* 中对应的 {0}、{1} 等占位字符，上面的网页执行结果如图 8.5 所示。



图 8.5　MessageTagDemo 项目的执行结果

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

8.1.5 <spring:transform> 标签

Spring 的 <spring:transform> 标签必须位于 <spring:bind> 标签之中，搭配 PropertyEditor 使用的话，可以将 Command 的属性值转换为您所要的字符串格式。

举个实际的例子，假设 Command 中储存了星期的数字表示，星期一使用 "1" 表示，星期二使用 "2" 表示……星期日使用 "0" 表示，然而您希望在填写表单时，可以在表单上显示的是 "MON"、"THU"、"WED" 等字样，您希望使用程序自动帮您完成转换。

可以先设计一个 WeekForm 类，收集请求参数值 "0" 到 "6"：

TransFormTagDemo — WeekForm.java

```
package onlyfun.caterpillar;

public class WeekForm {
    private String weekNumber;

    public void setWeekNumber(String weekNumber) {
        this.weekNumber = weekNumber;
    }

    public String getWeekNumber() {
        return weekNumber;
    }
}
```

weekNumber 储存的是 "1"、"2"、"3" 之类的请求参数值，接着可以设计一个 WeekPropertyEditor 类来转换这些参数值为 "MON"、"TUE" 到 "SUN"：

TransFormTagDemo — WeekPropertyEditor.java

```
package onlyfun.caterpillar;

import java.beans.PropertyEditorSupport;

public class WeekPropertyEditor extends PropertyEditorSupport {
    public void setAsText(String text) {
        if("MON".equals(text)) {
            setValue("1");
        }
        else if("TUE".equals(text)) {
            setValue("2");
        }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        else if("WED".equals(text)) {
            setValue("3");
        }
        else if("THU".equals(text)) {
            setValue("4");
        }
        else if("FRI".equals(text)) {
            setValue("5");
        }
        else if("SAT".equals(text)) {
            setValue("6");
        }
        else if("SUN".equals(text)) {
            setValue("0");
        }
        else {
            setValue("");
        }
    }

    public String getAsText() {
        if(this.getValue() != null
                && !(this.getValue() instanceof String)) {
            return "";
        }

        String text = (String) this.getValue();

        if("1".equals(text)) {
            return "MON";
        }
        else if("2".equals(text)) {
            return "TUE";
        }
        else if("3".equals(text)) {
            return "WED";
        }
        else if("4".equals(text)) {
            return "THU";
        }
        else if("5".equals(text)) {
            return "FRI";
        }
        else if("6".equals(text)) {
            return "SAT";
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
        else if("0".equals(text)) {
            return "SUN";
        }
        else {
            return text;
        }
    }
}
```

在设计 Controller 时，必须重新定义 initBinder() 方法以注册自制的 PropertyEditor，例如：

TransFormTagDemo　　WeekFormController.java

```
package onlyfun.caterpillar;

import org.springframework.web.servlet.
                            mvc.SimpleFormController;
import org.springframework.web.servlet.*;
import java.util.*;
import javax.servlet.http.HttpServletRequest;
import org.springframework.validation.BindException;
import org.springframework.web.
                        bind.ServletRequestDataBinder;

public class WeekFormController extends SimpleFormController {
    private Map refData;

    public WeekFormController() {
        setCommandClass(WeekForm.class);

        refData = new HashMap();

        List list = new ArrayList();
        for(int i = 0; i < 7; i++) {
            list.add(String.valueOf(i));
        }

        refData.put("weekNumbers", list);
    }

    protected ModelAndView onSubmit(Object command,
                BindException errors) throws Exception {
       WeekForm form = (WeekForm) command;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if(form.getWeekNumber() == null) {
            return new ModelAndView(this.getFormView());
        }
        else {
            return new ModelAndView(
                    this.getSuccessView(), errors.getModel());
        }
    }

    protected Map referenceData(
            HttpServletRequest request) throws Exception {
        return refData;
    }

    protected void initBinder(HttpServletRequest request,
            ServletRequestDataBinder binder) throws Exception {
        super.initBinder(request, binder);
        binder.registerCustomEditor(
                String.class, new WeekPropertyEditor());
    }
}
```

在构造 WeekFormController 时，会准备好 "0" 到 "6" 的字符串值，referenceData() 方法会回传包括这些字符串值的 Map 对象，以在页面上显示。由于 initBinder() 中注册了自制的 WeekPropertyEditor，再配合 <spring: transform> 标签，就可以在页面上显示 "SUN"、"MON" 等文字了，页面 *form.jsp* 的设计如下所示，当中搭配了 JSTL 来协助页面逻辑的编写：

TransFormTagDemo　　form.jsp

```
<%@page language="java" contentType="text/html;charset="BIG5"
<pageEncoding="BIG5"%>
<%@taglib prefix="c"
              uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="spring"
              uri="http://www.springframework.org/tags"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
    <head>
        <meta http-equiv="Content-Type"
             content="text/html; charset=BIG5">
        <title>JSP Page</title>
    </head>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <body>
    <spring:bind path="command.*">
        <font color="red">
            <c:out value="${status.errorMessage}"/>
        </font>
    </spring:bind>

    <form name="weekForm"
          action="/TransFormTagDemo/weekForm.do"
          method="POST">
        <spring:bind path="command.weekNumber">
            <select name="weekNumber">
                <c:forEach var="weekNumber"
                           items="${weekNumbers}" >
                    <spring:transform value="${weekNumber}"
                                      var="weekName"/>
                    <c:choose>

                    <c:when test="${status.value == weekName}">
                    <option value="<c:out value="${weekName}"/>"
                                    selected>
                    </c:when>
                    <c:otherwise>
                    <option value="<c:out value="${weekName}"/>">
                    </c:otherwise>

                    </c:choose>
                        <c:out value="${weekName}"/>
                    </option>
                </c:forEach>
            </select>
        </spring:bind>
        <input type="submit" value="Submit"/>
    </form>
    </body>
</html>
```

Bean 定义文件如下编写，由于使用了 JSTL，所以在 InternalResourceViewResolver 的 "viewClass" 的设置上可以使用 JstlView 类：

TransFormTagDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
        <property name="viewClass"
           value="org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <bean name="/weekForm.do"
         class="onlyfun.caterpillar.WeekFormController">
        <property name="formView" value="form"/>
        <property name="successView" value="success"/>
    </bean>

</beans>
```

如果您执行程序的话，当连到 *form.jsp* 时，会显示如图 8.6 所示的内容：

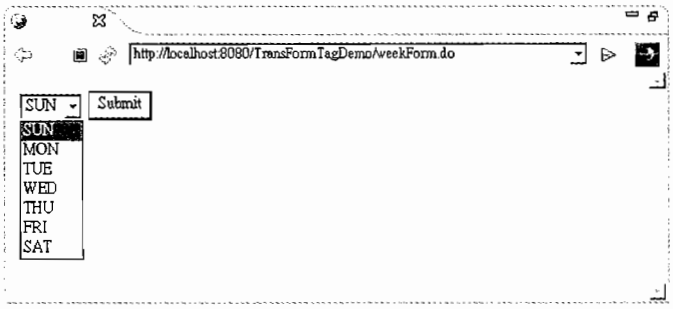


图 8.6　TransFormTagDemo 项目的执行结果

8.1.6　Spring 2.0 的 form 标签

Spring 2.0 中新增了一组 form 标签，包括 <form:form>、<form:input>、<form:password>等标签，方便您在设计表单网页时，可以提取 Command 对象的值并显示在网页中。想使用 Spring 2.0 新的 form 标签，要在 JSP 网页上加上：

```
<%@ taglib prefix="form"
          uri="http://www.springframework.org/tags/form" %>
```

以下使用 Spring 2.0 的 form 标签，改写 8.1.2 节的 SpringBindDemo 项目中的 *form.jsp*：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

FormTagDemo form.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@ taglib prefix="form"
          uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Login Form</title>
</head>
<body>
    <h1>登录表单</h1>
        请输入用户名称与密码: <p>
    <form:form>
        名称: <form:input path="username" /><br>
        密码: <form:password path="password" />
        <input type="submit" value="确定"/>
    </form:form>
</body>
</html>
```

以上的页面，在<form:form>标签的部分，将产生以下的 HTML 内容：

```
<form id="command" method="post" action="/FormTagDemo/login.do">
    名称: <input id="username" name="username" type="text" value=""/><br>
    密码: <input id="password" name="password" type="password" value=""/>
    <input type="submit" value="确定"/>
</form>
```

在<form:form>中，<form:input>用来产生文字输入字段，<form:password>用来产生密码输入字段，"path"属性的设置用来指定 Command 对象的属性，例如在这个范例中，<form:input>与<form:password>将分别参考至 LoginForm 实例的 username 与 password 属性。

<form:form>默认的"commandName"名称是"command"，如果您自行设置了 Controller 的 "commandName" 属性，则要在<form:form>标签的"commandName"上设置，例如"commandName"设置为"loginForm"：

```
...
    <form:form commandName="loginForm">
        名称: <form:input path="username" /><br>
        密码: <form:password path="password" />
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <input type="submit" value="确定"/>
    </form:form>
...
```

除了这里所介绍的<form:form>、<form:input>、<form:password>等标签之外，还有对应于 HTML 输入组件的标签，像是<form:checkbox>、<form:radiobutton>、<form:select>等，在使用上都很简单，一些范例可以直接参考 Spring 参考文件 13.9.2 The form tag 的内容。

8.2 其他 View 技术

您还可以使用其他的 View 层技术，或建立自己的 View 类，在这节示范 Tiles 作为 View 层技术，并以自制 Pdf 的结果画面来示范如何实现 View 类。

8.2.1 以 Tiles 为例

Tiles 是一个模版引擎(Template Engine)，可以让网页的配置(Layout)(像是使用 <table> 标签）与实际要呈现的数据分离，Tiles 提供一个版面管理机制，您可以在 Spring 的 View 层使用 Tiles 管理网页的版面配置，这有两个基本动作必须设置，一是设置 org.springframework.web.servlet.view.tiles.TilesView，二是设置 org.springframework.web.servlet.view.tiles.TilesConfigurer，例如：

TilesViewDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="viewResolver"
        class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
      <property name="viewClass" value="org.springframework.
                → web.servlet.view.tiles.TilesView"/>
    </bean>

    <bean id="tilesConfigurer"
        class="org.springframework.web.servlet.view.
                → tiles.TilesConfigurer">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <property name="definitions">
          <list>
            <value>/WEB-INF/tiles-defs.xml</value>
          </list>
        </property>
    </bean>

    <bean name="/welcome.do"
          class="org.springframework.web.servlet.
                    → mvc.ParameterizableViewController">
        <property name="viewName" value="welcome"/>
    </bean>

</beans>
```

注意到设置 TilesView 之后，您传回的 ModelAndView 中设置的 "view" 名称并不是资源的 URL 位置，而是在 Tiles 定义文件中所定义的名称，以上面的设置为例，ParameterizableViewController 的 "viewName" 设置为 "welcome"，则会到 Tiles 设置文件中找相关的 "welcome" 设置，Tiles 定义文件的读取位置可以由 TilesConfigurer 的 "definitions" 属性来设置，假设 Tiles 定义文件是这么设置的：

TilesViewDemo　　tiles-defs.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
 "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
 "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>
    <definition name=".myLayout"
                path="/WEB-INF/jsp/tiles/myLayout.jsp">
      <put name="title" value="Sample Page Title"/>
      <put name="header" value="/WEB-INF/jsp/tiles/header.jsp"/>
      <put name="menu" value="/WEB-INF/jsp/tiles/menu.jsp"/>
      <put name="footer" value="/WEB-INF/jsp/tiles/footer.jsp"/>
      <put name="body" value="/WEB-INF/jsp/tiles/body.jsp"/>
    </definition>

    <definition name="welcome" extends=".myLayout">
      <put name="title" value="Tiles 测试页面"/>
    </definition>
</tiles-definitions>
```

在定义文件中，首先定义一个 ".myLayout"，名称开头有个逗点，在命名惯例上表示一

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

个版面定义，而不是一个真正要呈现数据的网页定义，"welcome" 定义可以继承 ".myLayout" 的定义，并重新定义了 "title" 项目，在传回 View 的名称若为 "welcome" 时，实际上会读取的是定义文件中 "welcome" 的设置。

假设 *myLaout.jsp* 是这么编写的，当中使用表格来设计版面的配置：

TilesViewDemo　　myLayout.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="tiles"
          uri="http://jakarta.apache.org/struts/tags-tiles"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title><tiles:getAsString name="title"/></title>
</head>
<body>
    <table border="0" width="100%" cellspacing="5">
        <tr>
            <td colspan="2">
            <tiles:insert attribute="header"/>
            </td>
        </tr>
        <tr>
            <td width="140" valign="top">
            <tiles:insert attribute="menu"/>
            </td>
            <td valign="top" align="left">
            <tiles:insert attribute="body"/>
            </td>
        </tr>
        <tr>
            <td colspan="2">
            <tiles:insert attribute="footer"/>
            </td>
        </tr>
    </table>
</body>
</html>
```

header、menu、body、footer 等 JSP 页面简单地如下编写：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

TilesViewDemo header.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<center><h1>Tiles View 示范<h1> </center>
```

TilesViewDemo menu.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<li>教学文件</li>
<li>讨论区</li>
<li>我的 Blog</li>
<li>推荐网站</li>
```

TilesViewDemo body.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
第一个 Hello World 的出现是在 Brian Kernighan 写的 "A Tutorial Introduction to the
Language B" 书籍中（B 语言是 C 语言的前身），它用来将 Hello World 文字显示在电脑屏幕上，
自此之后，很多的程序语言教学文件或书籍上，已经无数次的将它当作第一个范例程序。
```

TilesViewDemo footer.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<center>版权所有(c)http://www.caterpillar.onlyfun.net/</center>
```

要使用 Tiles，必须要有 *commons-beanutils.jar*、*commonslogging.jar*、*commons-collections.jar*、*commons-digester.jar*、*struts.jar* 4 个 *.jar* 文件，这些 *.jar* 文件都可以在 Spring 下载文件中 *lib* 目录的 *jakarta-commons* 目录及 *struts* 目录下找到。

按照以上的设置，如果连接到 */welcome.do*，则会取得 Tiles 定义文件中 welcome 的定义，结果是显示以下的内容，如图 8.7 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。



图 8.7　TilesViewDemo 项目的执行结果

如果有一些数据是在每一个页面（例如在 Header 处）都要出现的，您可以在 *header.jsp* 中这么编写：

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<center><h1>Tiles View 示范<h1> </center>
<H1>${someData}</H1>
```

为了要能显示 ${someData} 数据，则必须在每一个 Controller 中设置属性值，例如：

```
...
Map model = new HashMap();
...
model.put("someData", data);
...
```

接着在 ModelAndView 传回时，设置 model 对象，每一个 Controller 中都必须加入这些描述，才会在每一个有 Tiles 设置的页面都显示 ${someData}。

不过还有个更简单的方法，您可以继承 org.springframework.web.servlet.view.tiles.ComponentControllerSupport 类，它继承 Tiles 的 ControllerSupport 类，您可以重新定义 ComponentControllerSupport 的 doPerform() 方法，例如：

```
...
public class HeaderControllerSupport
                extends ComponentControllerSupport {
    protected void doPerform(
                    ComponentContext componentContext,
                    HttpServletRequest request,
                    HttpServletResponse response)
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                                    throws Exception {
        ApplicationContext context = getApplicationContext();
            SomeData someData =
                  (SomeData) context.getBean("someData");

        componentContext.putAttribute("someData", someData);
    }
}
```

在 Tiles 定义文件中可以按如下设置，就可以让每一个页面都取得 someData 的值了：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
 "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
 "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>
    <definition name=".header"
                path="/WEB-INF/jsp/tiles/header.jsp"
                controllerClass="onlyfun.caterpillar.
                            → HeaderControllerSupport"/>

    <definition name=".myLayout"
                path="/WEB-INF/jsp/tiles/myLayout.jsp">
      <put name="title" value="Sample Page Title"/>
      <put name="header" value=".header"/>
      <put name="menu" value="/WEB-INF/jsp/tiles/menu.jsp"/>
      <put name="footer" value="/WEB-INF/jsp/tiles/footer.jsp"/>
      <put name="body" value="/WEB-INF/jsp/tiles/body.jsp" />
    </definition>

    <definition name="welcome" extends=".myLayout">
      <put name="title" value="Tiles 测试页面"/>
    </definition>
</tiles-definitions>
```

8.2.2 自制 View Class

您可以继承 org.springframework.web.servlet.view.AbstractView 自制的 View 类，以产生所需的 View 层技术，这里以产生 Pdf 的结果为例，可以直接继承 org.springframework.web.servlet.view.document.AbstractPdfView 类，并重新定义 buildPdfDocument() 方法，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

PdfViewDemo　　CustomPdfView.java

```
package onlyfun.caterpillar;

import com.lowagie.text.Document;
import com.lowagie.text.Paragraph;
import com.lowagie.text.pdf.PdfWriter;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.
               view.document.AbstractPdfView;

public class CustomPdfView extends AbstractPdfView {
   protected void buildPdfDocument(Map model,
                   Document document,
                   PdfWriter writer,
                   HttpServletRequest request,
                   HttpServletResponse response)
                                throws Exception {
       String header = (String) model.get("header");

       Paragraph headerParagraph =
                 new Paragraph(header);
       headerParagraph.font().setSize(20);
       document.add(headerParagraph);

       String content = (String) model.get("content");
       Paragraph contentParagraph =
                   new Paragraph(content);
       document.add(contentParagraph);

       String copyright = (String) model.get("copyright");
       Paragraph copyrightParagraph =
                        new Paragraph(copyright);
       copyrightParagraph.setAlignment(
                         Paragraph.ALIGN_BOTTOM);
       document.add(copyrightParagraph);
   }
}
```

假设在 Controller 中提供了 Model 数据对象：

PdfViewDemo　　PdfController.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class PdfController implements Controller {
    public ModelAndView handleRequest(HttpServletRequest req,
                    HttpServletResponse res) throws Exception {
        Map model = new HashMap();
        model.put("header", "Pdf View Test");
        model.put("content", "This is content for pdf view test.");
        model.put("copyright", "caterpillar.onlyfun.net");

        return new ModelAndView("customPdfView", model);
    }
}
```

而在 Bean 定义文件中是这么设计的：

PdfViewDemo　　mvc-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.BeanNameViewResolver"/>

    <bean id="customPdfView"
         class="onlyfun.caterpillar.CustomPdfView"/>

    <bean name="/pdf.do"
         class="onlyfun.caterpillar.PdfController"/>

</beans>
```

记得在*/WEB-INF/lib* 目录下包括 itext 的*.jar* 文件，则在连接上 */pdf.do* 时，会显示如图 8.8 所示的结果画面：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

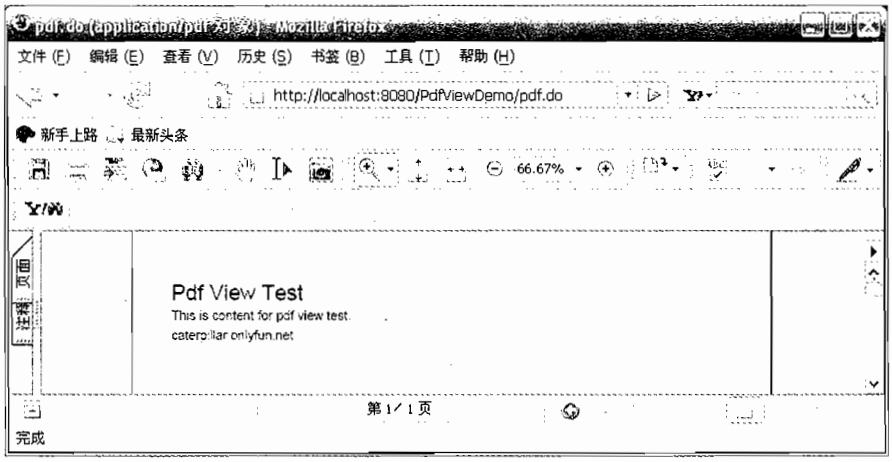


图 8.8　PdfViewDemo 项目的执行结果

8.3　整合 Struts 框架

Struts 是历史悠久的 Web MVC 框架，它是由 Craig McClanahan 于 2000 年 3 月发起的项目，在业界有相当多的项目使用 Struts 进行开发，即使 Struts 目前有许多设计不良的非议之声，但使用人员多，文件丰富为其优点，在这节中，将来介绍一下 Struts 的入门，以及如何将 Struts 与 Spring 整合在一起。

8.3.1　第一个 Struts 程序

第一个 Struts 程序的入门，是针对未接触过 Struts 的读者所安排的内容，目的是让您快速了解如何编写第一个 Struts 程序，以利在接下来的 8.3.2 节中实际练习 Spring 与 Struts 的整合范例，如果您已经了解 Struts 的编写方式，可以直接略过这个部分。

基于篇幅的限制，在这里不再介绍如何下载 Strtus 相关文件，而是直接使用 Spring 下载文件中 *lib* 目录下所提供的文件，如果想要下载最新的 Struts 实现，则可以至 *http://struts.apache.org* 进行下载（如图 8.9 所示）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

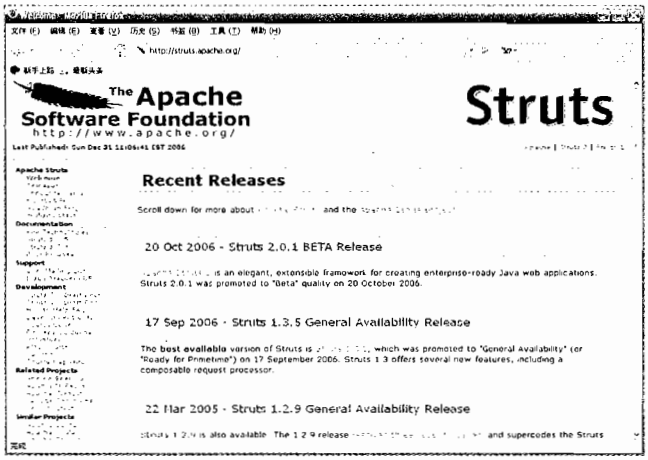


图 8.9 Struts 官方网站

对于第一个 Struts 程序来说，需要以下的*.jar* 文件，可以分别在 Spring 下载文件的 *lib* 目录下的 *jakarta-commons* 与 *struts* 目录中找到这些 *.jar* 文件，如果您先前下载的 Spring 是相依赖版本，也可以在 *lib/struts* 与 *lib/jakarta-commons* 目录下找到这些文件：

- struts.jar
- commons-beanutils.jar
- commons-digester.jar
- commons-collections.jar
- commons-logging.jar

在 Struts 中，担任前端控制器（Front Controller）的是 `org.apache.struts.action.ActionServlet` 类，您必须在 *web.xml* 中加以定义，并指定 Struts 设置文件的位置与名称，例如：

FirstStrutsDemo　　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  → http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
   <session-config>
      <session-timeout>
         30
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        </session-timeout>
    </session-config>

    <!-- Standard Action Servlet Configuration -->
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
            org.apache.struts.action.ActionServlet
        </servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>
                /WEB-INF/struts-config.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Standard Action Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
</web-app>
```

ActionServlet 的 "config" 属性用来设置 Struts 设置文件的位置与名称，在以上的设置中，所有对 *.do 的请求，都会交由 ActionServlet 来分配给控制对象处理。

在 Struts 中，控制对象的实现是继承 org.apache.struts.action.Action 类，并重新定义 execute() 方法来完成，例如在以下的实现中，将取得用户的 "user" 请求参数值，并设置给 Map 类型的 Model 对象：

FirstStrutsDemo　　HelloAction.java

```
package onlyfun.caterpillar;

import java.util.*;
import javax.servlet.http.*;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class HelloAction extends Action {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public ActionForward execute(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response)
                                             throws Exception {
        String username = request.getParameter("user");

        Map model = new HashMap();

        if(username != null) {
            model.put("username", username);
        }
        else {
            model.put("username", "nobody");
        }

        request.setAttribute("userInfo", model);

        return mapping.findForward("helloUser");
    }
}
```

在请求处理完成之后，必须由 org.apache.struts.action.ActionMapping 来查询呈现页面的地址，ActionMapping 是 Struts 定义文件中的 <action-mappings> 定义的实例代表，来看一下 Struts 定义文件是如何编写的：

FirstStrutsDemo　　struts-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts-config PUBLIC
 "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
 "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
 <struts-config>
    <action-mappings>
        <action
            path="/hello"
            type="onlyfun.caterpillar.HelloAction">
            <forward
                name="helloUser"
                path="/WEB-INF/jsp/hello.jsp"/>
        </action>
    </action-mappings>
</struts-config>
```

在 Struts 定义文件中，当<action> 标签的 "path" 属性设置的请求路径为 *hello.do* 时，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

ActionServlet 将交给 HelloAction 来处理用户的请求，而当 <forward> 标签设置使用 ActionMapping 对象的 findForward() 方法查询名称时，要告知的下一个呈现页面的地址，例如在 HelloAction 中 findForward() 设置的名称是 "helloUser"，因此请求处理完毕后，将由 */WEB-INF/jsp/hello.jsp* 来呈现结果画面，*hello.jsp* 的编写如下所示：

FirstStrutsDemo　　hello.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>Struts Demo</title>
</head>
<body>
    <H1>Hello, ${userInfo["username"]} !</H1>
</body>
</html>
```

hello.jsp 网页只是将 request 中 Map 对象的 "username" 数据显示出来，一个执行画面如图 8.10 所示。

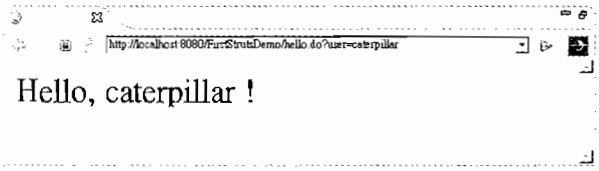


图 8.10　FirstStrutsDemo 项目的执行结果

良葛格的话匣子<<<

有关于 Struts 更详尽的介绍，可以参考我网站上的 Struts 文件：

http://caterpillar.onlyfun.net/Gossip/Struts/StrutsGossip.htm

8.3.2　在 Struts 中整合 Spring

Struts 要与 Spring 结合使用，主要的方法就是让 Struts 知道 Spring 的存在，以让 Spring 管理相关的组件，避免在程序中直接编写组件的依赖关系建立。

首先要在 Struts 的 *struts-config.xml* 中使用 <plug-in> 标签注册 org.springframework.web.struts.ContextLoaderPlugIn：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
...
    <plug-in className="org.springframework.web.
              → struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation"
                  value="/WEB-INF/beans-config.xml, /WEB-INF/..."/>
    </plug-in>
...
```

有几个方法可以取得 Spring 所管理的 Bean，方法之一是 Struts 的 Action 改继承 org.springframework.web.struts.ActionSupport，这个类是 Struts 的 Action 抽象类的实现，可以使用它的 getWebApplicationContext() 来取得 ApplicationContext 的实例，然后进一步取得 Spring 容器所管理的 Bean 实例，例如：

```
...
public class SomeAction extends ActionSupport {
    public ActionForward execute(ActionMapping mapping,
                            ActionForm form,
                            HttpServletRequest req,
                            HttpServletResponse res)
                                        throws Exception {
        ApplicationContext context =
                                getWebApplicationContext();

        SomeBean bean = (SomeBean) context.getBean("some");
        ...
        return mapping.findForward("view");
    }
}
...
```

这种方式符合 Struts 的使用习惯，通过继承来实现 Action，并可得到 Spring 管理组件关系的好处，但坏处就是 Spring 与 Struts 的 API 耦合在一起，而且 Action 中包括了取得相依赖对象的逻辑，这并没有善用 Spring 管理依赖注入的好处。

另一个方式是让 Struts 的 Action 直接继承它自己的抽象 Action 类，但让 Spring 来管理 Struts 的 Action 对象，让 Action 对象也成为 Spring 管理的 Bean，这么一来就可以直接使用依赖注入的方式注入相依赖对象，例如可以重新修改一下 FirstStrutsDemo 项目中的 HelloAction 类：

StrutsSpringDemo　　HelloAction.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import java.util.*;
import javax.servlet.http.*;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class HelloAction extends Action {
    private UserChecker userChecker;

    public ActionForward execute(
                        ActionMapping mapping,
                        ActionForm form,
                        HttpServletRequest request,
                        HttpServletResponse response)
                            throws Exception {
        String username = request.getParameter("user");
        username = this.getUserChecker().check(username);

        Map model = new HashMap();
        model.put("username", username);
        request.setAttribute("userInfo", model);

        return mapping.findForward("helloUser");
    }

    public UserChecker getUserChecker() {
        return userChecker;
    }

    public void setUserChecker(UserChecker userChecker) {
        this.userChecker = userChecker;
    }
}
```

其中 UserChecker 类的定义如下所示：

StrutsSpringDemo　　UserChecker.java

```
package onlyfun.caterpillar;

public class UserChecker {
    public String check(String username) {
        if(username != null) {
            return username;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
        else {
            return "guest";
        }
    }
}
```

UserChecker 模拟业务层的权限检查对象，您要在 Struts 的 Action 中使用到 UserChecker 的实例，这可以交由 Spring 来为您作依赖注入，可以将 HelloAction 与 UserChecker 的实例都交给 Spring 管理，例如在 Bean 定义文件中这么定义：

StrutsSpringDemo — beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="userChecker"
         class="onlyfun.caterpillar.UserChecker"/>

    <bean name="/hello"
         class="onlyfun.caterpillar.HelloAction">
        <property name="userChecker" ref="userChecker"/>
    </bean>

</beans>
```

Action 的实例现在已纳入 Spring 的管理，那么 Struts 在分配请求时，要有一个中间代理机制，在请求要分配至指定的 Action 之前，先分配至代理对象，由代理对象通知 Spring 所管理的 Action 实例来处理请求，并将处理结果返回给代理对象，再由代理对象返回给 Struts，这可以在 strutsconfig.xml 中使用 org.springframework.web.struts.DelegatingActionProxy 的实例来作代理，例如：

StrutsSpringDemo — struts-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts-config PUBLIC
 "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
 "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
 <struts-config>
    <action-mappings>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <action
            path="/hello"
            type="org.springframework.web.
                    → struts.DelegatingActionProxy">
            <forward
                name="helloUser"
                path="/WEB-INF/jsp/hello.jsp"/>
        </action>
    </action-mappings>

    <plug-in className="org.springframework.web.
                        → struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation"
                    value="/WEB-INF/beans-config.xml"/>
    </plug-in>
</struts-config>
```

在定义文件中使用<plug-in> 标签加入 ContextLoaderPlugIn，并指定了 Spring 的 Bean 定义文件之位置与名称。

注意若将 *beans-config.xml* 中 HelloAction 的 "name" 属性设置为 "/hello"，则 <action> 中的 "path" 属性也必须设置为 "/hello"，DelegatingActionProxy 是借着这个来找到 Action 实例的，这个方法的缺点就是要花功夫在两个定义文件的名称对比上。StrutsSpringDemo 项目到这里已修改完成，执行的结果画面可以参考一下图 8.10 的画面。

您可以使用 Spring 的 org.springframework.web.struts.DelegatingRequest-Processor 来取代 Struts 的 RequestProcessor，在 *strutsconfig.xml* 中定义：

```
...
<plug-in className="org.springframework.web.
            → struts.ContextLoaderPlugIn">
    <set-property property="contextConfigLocation"
                value="/WEB-INF/beans-config.xml"/>
</plug-in>

<controller processorClass="org.springframework.web.
            → struts.DelegatingRequestProcessor"/>
...
```

这一次，可以直接将 Struts 的 Action 类名称写在 *struts-config.xml* 上，看来会比较直观，例如：

```
...
<action path="/hello"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            type="onlyfun.caterpillar.HelloAction"/>
...
```

事实上并不会使用到 "type" 属性的设置，编写出来只是为了比较清楚 Action 使用了哪一个类，简洁的写法只要这样就可以了：

```
...
<action path="/someAction"/>
...
```

同样的，当请求分配时，会由代理对象将请求分配至 Bean 定义文件中具有相同名称（/hello）的 Action 实例。

8.4 整合 JSF 框架

JavaServer Faces 为 JCP（Java Community Process）所提出的一个技术标准，综合各家厂商现有的技术特点，提供 Web 的 View 层可重复使用的 UI（User Interface）组件，并支持整合开发环境以拖曳的方式快速构造用户接口，并提供类似桌面窗口应用程序的开发模型，让习惯桌面窗口程序开发的人员，也可以快速上手 Web 程序开发，在这节中，将来介绍如何开发第一个 JSF 应用程序，以及如何整合 JSF 与 Spring。

8.4.1 第一个 JSF 程序

第一个 JSF 程序的入门，是针对未接触过 JSF 的读者所安排的内容，目的是让您快速了解如何编写第一个 JSF 程序，以利在接下来的 8.4.2 节中，实际练习 Spring 与 JSF 的整合范例，如果您已经了解 JSF 的编写方式，可以直接略过这个部分。

Java EE 包括了 JSF，基于篇幅的限制，在这里不再介绍如何下载 JSF 实现的相关文件，JSF 的实现参考文件可以至 Java 官方网站 *http://java.sun.com/j2ee/javaserverfaces/download.html* 进行下载，如图 8.11 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

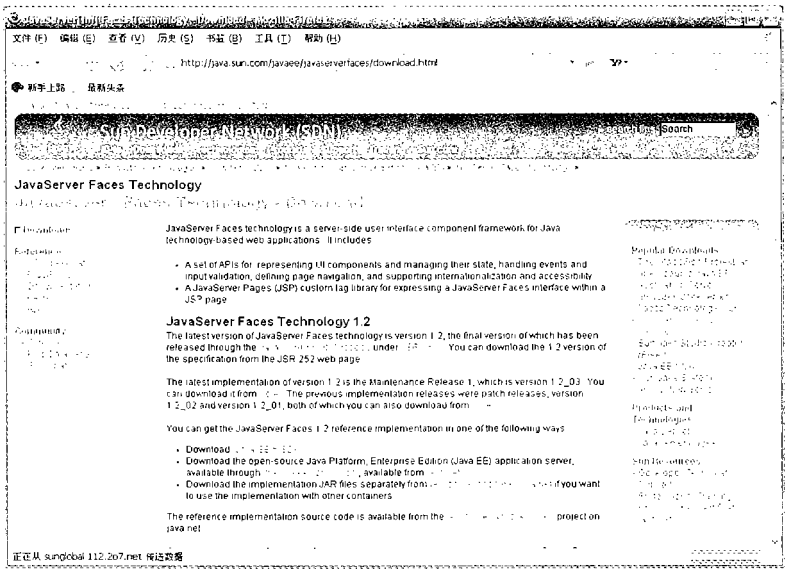


图 8.11　JSF 实现参考的下载网页

良葛格的话匣子<<<

JSF 是一个技术标准，各个厂商可以根据标准的规范来进行自己的实现版本，Sun 网站上提供的是 Sun 公司的实现参考。

对于要实现的第一个 JSF 程序，将使用到 *jsf-impl.jar* 与 *jsf-api.jar*，另外因 JSF 需要使用到 JSTL 的 API，因而需要 JSTL 的 *jstl.jar* 与 *standard.jar*，除此之外，还需要 *commons-digester.jar*、*commons-collections.jar*、*commons-beanutils.jar* 等文件。总计所需要的 *.jar* 如下所示：

- jsf-impl.jar
- jsf-api.jar
- jstl.jar
- standard.jar
- commons-digester.jar
- commons-collections.jar
- commons-beanutils.jar

JSF 也是采用 Web MVC 架构，使用 JSF 时担任前端控制器（Front controller）的是 `javax.faces.webapp.FacesServlet` 类，您要在 *web.xml* 定义它，例如：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

FirstJSFDemo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  → http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>
            javax.faces.webapp.FacesServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
</web-app>
```

在上面的定义中，将所有的 ".faces"请求交由 FaceServlet 来处理，FaceServlet 会唤起相对于请求网址的 *.jsp* 网页，例如若请求是 */index.faces*，则实际上会唤起 */index.jsp* 网页。完成以上的配置，就可以开始使用 JSF 了，先来编写一个简单的 Bean 类：

FirstJSFDemo User.java

```
package onlyfun.caterpillar;

public class User {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

这个 User 类只是个简单的 JavaBean，用以储存用户的请求参数值。JSF 与 Struts 虽然都是 Web MVC 架构的实现，但两者所偏重的并不相同，Struts 着重的是控制对象的设计；JSF 则着重于页面流程的设计，也就是定义在何种条件成立下，上一个页面与下一个页面之间如何连结。可以在 *faces-config.xml* 中定义页面流程，例如：

FirstJSFDemo　　faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
    <navigation-rule>
        <from-view-id>/pages/index.jsp</from-view-id>
        <navigation-case>
            <from-outcome>login</from-outcome>
            <to-view-id>/pages/welcome.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>user</managed-bean-name>
            <managed-bean-class>
                onlyfun.caterpillar.User
             </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

在 <navigation-rule> 标签中定义了页面流程，当页面流程上的一个页面是来自 <from-view-id> 标签中指定的页面，并且请求中包括了 <navigation-case> 标签中的 <from-outcome> 标签所指定的 "login" 值时，则会将请求导向至 <to-view-id> 标签中所指定的页面。

在上面的定义中也可以看到，JSF 也协助 Bean 对象的管理，只是功能比 Spring 容器少了许多；在 <managed-bean> 标签中可以管理 Bean 对象，可设置 Bean 对象的存活范围为 "session"，也就是在用户开启浏览器与 Web 程序互动的会话阶段，Bean 都可以存在作用。

接下来要告诉网页设计人员的信息是，他们可以使用的 Bean 名称，也就是 <managed-bean-name> 标签中设置的 "user" 名称，以及上面所定义的页面流程，在使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

页面组件时需要设置这些值，例如网页设计人员会编写以下的 JSP 网页：

FirstJSFDemo index.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
<title>第一个 JSF 程序</title>
</head>
<body>
    <f:view>
        <h:form>
            <h3>请输入您的名称</h3>
            名称: <h:inputText value="#{user.name}"/><p>
            <h:commandButton value="送出" action="login"/>
        </h:form>
    </f:view>
</body>
</html>
```

在上面的网页中使用了 JSF 的 "core" 与 "html" 标签库，"core" 是有关于 UI 组件的处理，而 "html" 则是有关于 HTML 的高级标签。

开始使用 JSF 组件时，这些组件一定要在 <f:view> 与 </f:view> 标签之间；"html" 标签库中几乎都是与 HTML 标签相关的高级标签，<h:form> 会产生一个表单，使用 <h:inputText> 来显示 *faces-config.xml* 定义文件中 "user" 这个 Bean 对象的 "name" 属性，而 <h:commandButton> 会产生一个提交按钮，在 "action" 属性中之指定将根据 *faces-config.xml* 中定义的 "login" 页面流程中前往 *welcome.jsp* 页面，其设计如下：

FirstJSFDemo welcome.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=BIG5">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<title>第一个 JSF 程序</title>
</head>
<body>
    <f:view>
        <h:outputText value="#{user.name}"/> 您好!
        <h3>欢迎使用 JavaServer Faces! </h3>
    </f:view>
</body>
</html>
```

<h:outputText> 标签可以取得 "user" 这个 Bean 对象的 "name" 属性，这个属性将包括先前 *index.jsp* 中所送出的请求参数值，而在这个网页中则将之显示出来，首先来看一下 *index.jsp* 的执行结果，如图 8.12 所示。

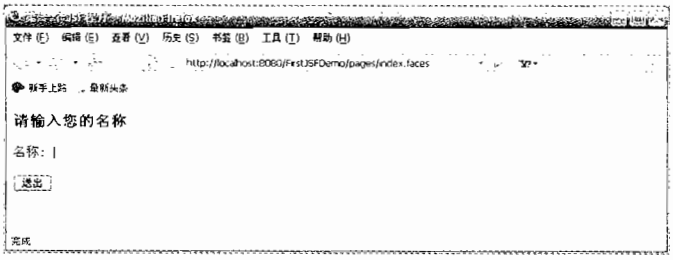


图 8.12　FirstJSFDemo 项目的 index.jsp 网页执行结果

在名称字段中填写完毕并点击送出按钮后，可以看到如图 8.13 所示的结果画面。

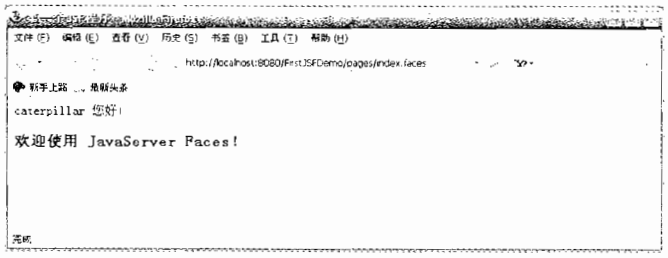


图 8.13　FirstJSFDemo 项目的 welcome.jsp 网页执行结果

如果想体验一下 JSF 在 IDE 中所视即所得的开发方式，可以至 *http://developers.sun.com/prodtech/javatools/jscreator/index.jsp* 下载 Sun Java Studio Creator，Sun 公司现已提供该开发工具免费下载，图 8.14 为 Sun Java Studio Creator 的所视即所得网页组件编辑画面。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

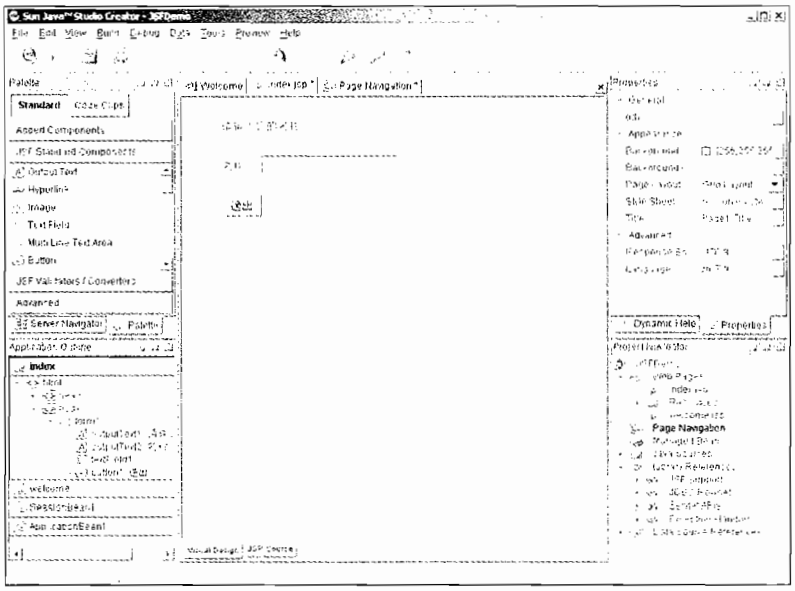


图 8.14 Sun Java Studio Creator 的所视即所得网页组件编辑

而在页面流程的编辑上，Sun Java Studio Creator 也提供可视化的设计方式，如图 8.15 所示。

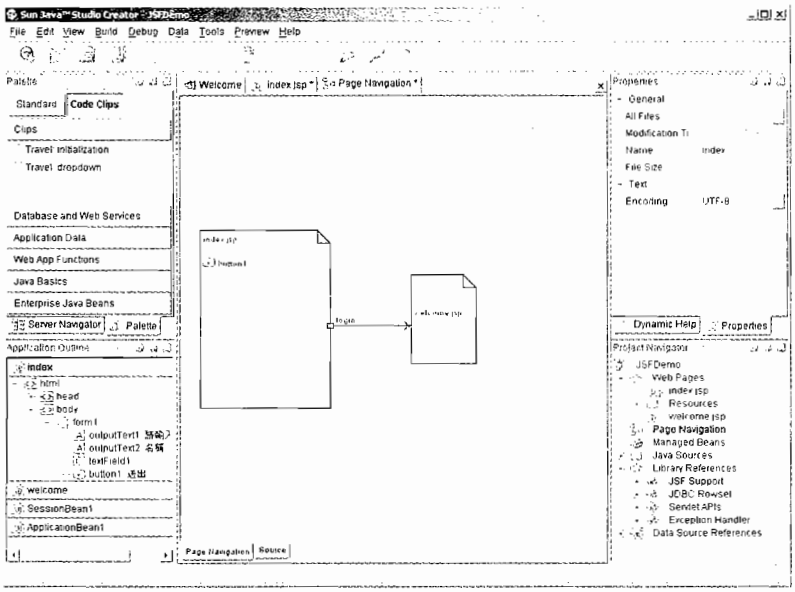


图 8.15 Sun Java Studio Creator 可视化的页面流程编辑

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

良葛格的话匣子<<<

有兴趣的话，可以在我的网站上找到更多有关于 JSF 的介绍：

http://caterpillar.onlyfun.net/Gossip/JSF/JavaServerFaces.htm

8.4.2　在 JSF 中整合 Spring

JSF 本身对于 Bean 的管理有依赖注入的功能，不过 Spring 的 IoC 容器提供更多的功能，另一方面 Spring 还有 AOP 框架，在事务处理上也可提供协助。

JSF 与 Spring 的结合，主要目的就是让 Spring 的 Bean 名称，可以被 JSF 视作 `<managed-bean-name>` 标签上的名称来使用。也就是说当 JSF 定义文件中根据名称要求 Bean 实例时，若 JSF 在自己的 Context 中找不到名称，可以尝试到 Spring 的 Context 中去寻找。

来改写一下 FirstJSFDemo 项目，以示范 JSF 与 Spring 如何结合运行，首先设计一个简单的 `UserChanger` 类：

JSFSpringDemo　　UserChanger.java

```
package onlyfun.caterpillar;

public class UserChanger {
    public String getChangedName(String name) {
        return name.toUpperCase();
    }
}
```

这个类会将传入的名称改成大写，用来模拟业务层的一个对象，接着来改写一下 `User` 类：

JSFSpringDemo　　User.java

```
package onlyfun.caterpillar;

public class User {
    private String name;
    private UserChanger userChanger;

    public void setName(String name) {
        this.name = getUserChanger().getChangedName(name);
    }

    public String getName() {
        return name;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    public void setUserChanger(UserChanger userChanger) {
        this.userChanger = userChanger;
    }

    public UserChanger getUserChanger() {
        return userChanger;
    }
}
```

User 类中接受 UserChecker 类的实例注入，用户送出的名称会被 UserChecker 类的实例改为大写，可以使用 org.springframework.web.jsf.DelegatingVariableResolver 类让 Spring 的 Bean 名称可以在 JSF 中使用。可以在 JSF 的 *faces-config.xml* 中设置：

JSFSpringDemo　　faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
    <application>
        <variable-resolver>
       org.springframework.web.jsf.DelegatingVariableResolver
        </variable-resolver>
    </application>
    <navigation-rule>
        <from-view-id>/pages/index.jsp</from-view-id>
        <navigation-case>
            <from-outcome>login</from-outcome>
            <to-view-id>/pages/welcome.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>user</managed-bean-name>
        <managed-bean-class>
            onlyfun.caterpillar.User
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
        <managed-property>
          <property-name>userChanger</property-name>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            <value>#{userChanger}</value>
        </managed-property>
    </managed-bean>
</faces-config>
```

实际上在 JSF 的定义文件中并没有定义"userChanger"的 Bean 实例，当 JSF 找不到"userChanger"时，会使用 DelegatingVariableResolver 对象在 Spring 的 Context 中寻找同名的 Bean 实例，假设 Bean 定义文件是这么编写的：

JSFSpringDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="userChanger"
          class="onlyfun.caterpillar.UserChanger"/>

</beans>
```

为了让 DelegatingVariableResolver 对象知道 Bean 定义文件的名称与位置，必须在 *web.xml* 中使用 org.springframework.web.context.ContextLoaderListener 类，并使用 <context-param> 标签指定 Bean 定义文件的名称与位置，例如：

JSFSpringDemo　　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 → http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

    <listener>
        <listener-class>
            org.springframework.web.
                    → context.ContextLoaderListener
        </listener-class>
    </listener>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/beans-config.xml</param-value>
    </context-param>

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>
            javax.faces.webapp.FacesServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
</web-app>
```

在不支持 Listener 设置的容器上（例如 Servlet 2.2 以及更早的版本），可以使用 `org.springframework.web.context.ContextLoaderServlet` 来取代上面的 Listener 设置，例如：

```
...
<servlet>
    <servlet-name>demoConfig</servlet-name>
    <servlet-class>
    org.springframework.web.
          → context.ContextLoaderServlet
    </servlet-class>
</servlet>
...
```

现在可以运行一下程序，看一下执行的结果中送出的用户名称是否会被改为大写，假设送出的用户名称为 "caterpillar"，执行时的参考画面如图 8.16 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

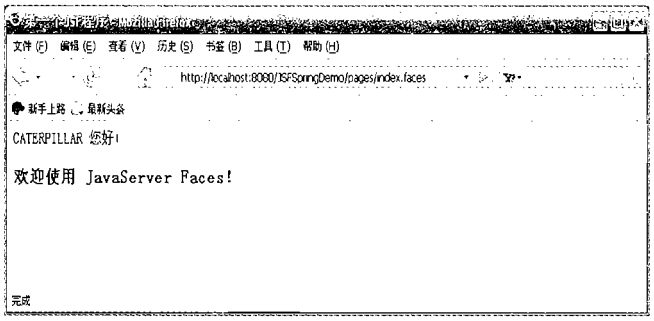


图 8.16　JSFSpringDemo 项目的执行结果

8.5　接下来的主题

Spring 对 Web 层的支持，暂且说明到这里，在 Spring 的参考手册中，还有更多关于 Web 层整合方案的说明，在下一个章节，将来介绍一下 Spring 于远程服务、邮件服务、排程服务等所提供的一致性封装模型，基于 Spring 的封装，无论底层由什么方案实现，在 Spring 中的使用方式都是一致的。

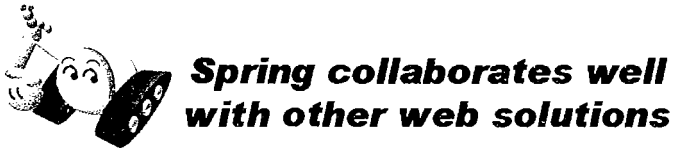


图 8.17　Spring 可以与其他的 Web 方案整合使用

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第

API 封装

章

对于一些企业服务 API，Spring 不提供直接的解决方案，而是提供一层抽象封装，在这样的封装下，提供了简化且一致的方式，让您在使用一些 API 或组件服务时更加简单。

这个章节中将介绍远程服务、邮件服务与任务计划服务等在 Spring 中的使用方式。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

9.1　远程

远程服务让本地的对象在使用服务器端服务时，就如同使用本地服务一样，客户端不需接触到网络连接等细节，对于各种不同的远程技术，Spring 提供了一致的使用方式，即使底层所采用的远程服务技术各不相同，在 Spring 中运用它们的方式却是一致的，这节将介绍一下 RMI、Hessian、Burlap 以及 Http Invoker 的使用。

9.1.1　RMI

RMI（Remote Method Invocation）是从 JDK 1.1 开始就出现的 API，它让客户端在使用远程对象提供的服务时，就如同使用本地对象一样，然而在使用 RMI 时必须有一连串的手续，如服务接口在定义时必须继承 `java.rmi.Remote` 接口、服务 Server 在实现时必须继承 `java.rmi.UnicastRemoteObject` 类、必须使用 `rmic` 指令产生 stub 与 skeleton 等，设置上的手续繁杂。

您可以在 Spring 中通过 `org.springframework.remoting.rmi.RmiServiceExporter` 简化使用 RMI 的手续，来看看实际例子，了解 Spring 在 RMI 上的使用与简化，首先看一下 RMI 服务器端的编写，然后定义一个服务对象的接口：

RMIServerDemo　　ISomeService.java

```
package onlyfun.caterpillar;

public interface ISomeService {
    public String doSomeService(String some);
    public int doOtherService(int other);
}
```

服务对象的接口不用继承 `java.rmi.Remote` 接口，而且在实现 `ISomeService` 时也不用继承 `java.rmi.UnicastRemoteObject` 类，例如：

RMIServerDemo　　SomeServiceImpl.java

```
package onlyfun.caterpillar;

public class SomeServiceImpl implements ISomeService {
    public String doSomeService(String some) {
        return some + " is processed";
    }

    public int doOtherService(int other) {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        return ++other;
    }
}
```

这个实现只是个简单的示范，两个方法都只是传回一个已经修改过的值，接下来只要在Bean定义文件中定义，让Spring管理、生成Bean实例，如此即可注册、启动RMI服务，例如：

RMIServerDemo　　rmi-server.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someService"
         class="onlyfun.caterpillar.SomeServiceImpl"/>

    <bean id="serviceExporter"
         class="org.springframework.remoting.
                → rmi.RmiServiceExporter">
        <property name="service" ref="someService"/>
        <property name="serviceName" value="SomeService"/>
        <property name="serviceInterface"
                value="onlyfun.caterpillar.ISomeService"/>
    </bean>

</beans>
```

只要告诉 org.springframework.remoting.rmi.RmiServiceExporter 服务对象、名称（注意在 "serviceName" 属性上设置为 "SomeService"）与要代理的接口，在 Spring读取定义文件并生成Bean实例后，RMI服务就会启动，来编写一个简单的 RMIServer 类，以启动RMI服务：

RMIServerDemo　　RMIServer.java

```
package onlyfun.caterpillar;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import org.springframework.context.ApplicationContext;
import org.springframework.context.
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
      support.ClassPathXmlApplicationContext;
import org.springframework.remoting.rmi.RmiServiceExporter;

public class RMIServer {
    public static void main(String[] args)
                        throws IOException {
     ApplicationContext context =
         new ClassPathXmlApplicationContext("rmi-server.xml");

        System.out.println("启动 RMI Server..");
        System.out.println("请输入 exit 关闭 Server: ");

        BufferedReader reader =
            new BufferedReader(new InputStreamReader(System.in));

        while(true) {
            if(reader.readLine().equals("exit")) {
                break;
            }
        }

        RmiServiceExporter rmiServiceExporter =
           (RmiServiceExporter) context.getBean("serviceExporter");
        rmiServiceExporter.destroy();
    }
}
```

在运行上面的程序之后，RMI 服务就会启动，Spring 会自动使用另一个线程来执行 RMI 服务，所以不用关心线程的处理问题，您可以输入 "exit" 直接离开程序，接着来看一下，如何实现一个 RMI 客户端以向 RMI 服务器要求服务，首先要记得的是，客户端是依赖于抽象的接口，也就是说先前 ISomeService 接口的 *.class* 文件也必须在客户端有一份。

当客户端需要 RMI 服务时，只要通过 org.springframework.remoting.rmi.RmiProxyFactoryBean，并告知服务的 URL（对应至先前设置的"SomeService"名称）、代理的接口即可，在编写程序时就好像在使用本地端管理的服务一样，例如 Bean 定义文件可以如下编写：

RMIClientDemo　　rmi-client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someServiceProxy"
          class="org.springframework.remoting.
                  → rmi.RmiProxyFactoryBean">
        <property name="serviceUrl"
                  value="rmi://localhost/SomeService"/>
        <property name="serviceInterface"
                  value="onlyfun.caterpillar.ISomeService"/>
    </bean>

</beans>
```

注意到 "serviceUrl" 属性的设置，它以 "rmi://" 开头，接着指定服务器地址与服务名称。来编写一个简单的客户端程序以使用 RMI 服务器上的服务：

RMIClientDemo　　RMIClient.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
            support.ClassPathXmlApplicationContext;

public class RMIClient {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "rmi-client.xml");

        ISomeService service =
            (ISomeService) context.getBean("someServiceProxy");

        String result1 = service.doSomeService("Some request");
        System.out.println(result1);

        int result2 = service.doOtherService(1);
        System.out.println(result2);
    }
}
```

在程序的实现中，完全不需要处理到有关服务连接的种种细节，代理对象会自动帮您完成这些细节，单从程序来看，根本不会注意到正在取得远程服务器上的服务，执行的结果如图 9.1 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

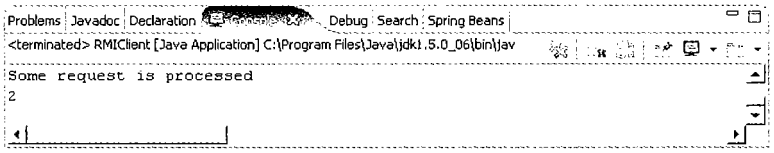


图 9.1　RMIClientDemo 项目的执行结果

9.1.2　Hessian、Burlap

Hessian、Burlap 由 Caucho Technology（*http://www.caucho.com/*）提出，它们是通过 HTTP 实现的远程服务。

Hessian 是使用 HTTP 将对象以中性的二进制消息进行传送，而不像 RMI 使用 Java 的序列化格式，由于二进制消息是中性的，因此不受限于某种程序语言所实现的客户端或服务器端，二进制数据在传输时所需的带宽较小是其优点。

Burlap 则是以 XML 文件格式传送对象，XML 文件有较高的可读性，只要应用程序可以剖析 XML 文件就可以解读所接收的消息，当然也不受限于某种语言所实现的客户端与服务器端。

在 Spring 中使用 Hessian 及 Burlap 的方法是类似的，由于 Hessian、Burlap 是通过 HTTP 传送的，所以在使用时要搭配 Spring Web 框架，也就是使用 `DispatcherServlet`，举个实际的例子来示范如何使用 Hessian，以 9.1.1 节中介绍的 RMI 例子来说，可以直接使用已编写好的 `ISomeService`、`SomeServiceImpl`，而服务器端要在 *web.xml* 中配置 `DispatcherServlet`：

HessianServerDemo　　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
          <param-name>contextConfigLocation</param-name>
          <param-value>/WEB-INF/service-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>*.service</url-pattern>
    </servlet-mapping>
</web-app>
```

在 Hessian 的服务器端，使用 `org.springframework.remoting.caucho.HessianServiceExporter` 来发布服务：

HessianServerDemo　　service-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                  → handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/some.service">serviceExporter</prop>
            </props>
        </property>
    </bean>

    <bean id="someService"
         class="onlyfun.caterpillar.SomeServiceImpl"/>

    <bean id="serviceExporter"
         class="org.springframework.remoting.
                  → caucho.HessianServiceExporter">
        <property name="service" ref="someService"/>
        <property name="serviceInterface"
                value="onlyfun.caterpillar.ISomeService"/>
    </bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
</beans>
```

注意到在 SimpleUrlHandlerMapping 的设置上，请求"some.service"的会分配给"serviceExporter"，在这里不用注册服务名称，也就是没有 "serviceName" 属性，Hessian、Burlap 不需要这属性，如果使用的是 Burlap，则只要在"serviceExporter"的"class" 属性改用 org.springframework. remoting.caucho.BurlapServiceExporter 类即可。

接下来只要启动 Servlet 容器，加载以上设计的 Web 应用程序，之后 Hessian 服务器端就会启动了，记得要在 *lib* 目录中加入 Hessian 所需的 API 类之 *.jar* 文件，这可以使用 Spring 下载文件中 *lib* 目录下 *caucho* 目录的 *hessian-3.0.20.jar* 文件，如果使用 Burlap 的话，要记得加入 Burlap API 所需的 *.jar* 文件。

Hessian 客户端的编写则可以使用先前编写的 RMIClientDemo 项目来改写，事实上只要修改一下 Bean 定义文件即可，例如：

HessianClientDemo　　hessian-client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someServiceProxy"
         class="org.springframework.remoting.
                → caucho.HessianProxyFactoryBean">
       <property name="serviceUrl" value="http://localhost:8080/
                → HessianServerDemo/some.service"/>
       <property name="serviceInterface"
          value="onlyfun.caterpillar.ISomeService"/>
    </bean>

</beans>
```

注意到 "serviceUrl" 属性的设置，它是个标准的 HTTP 请求地址，来编写个简单的客户端程序以使用 Hessian 服务器上的服务：

HessianClientDemo　　HessianClient.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.
            support.ClassPathXmlApplicationContext;

public class HessianClient {
   public static void main(String[] args) {
      ApplicationContext context =
         new ClassPathXmlApplicationContext(
               "hessian-client.xml");

      ISomeService service =
         (ISomeService) context.getBean("someServiceProxy");
      String result1 = service.doSomeService("Some request");
      System.out.println(result1);

      int result2 = service.doOtherService(1);
      System.out.println(result2);
   }
}
```

同样必须记得，要加入包括 Hessian API 的 *.jar* 文件，执行的结果与 RMIClientDemo 项目是相同的，可以参考图 9.1 的画面，如果要使用 Burlap，则只要在设置上改用 `org.springframework.remoting.caucho.BurlapProxyFactoryBean` 即可，当然要记得必须加入 Burlap API 所需的 *.jar* 文件。

9.1.3 Http Invoker

Http Invoker 使用 HTTP 传送对象，传送时须使用 Java 的序列化机制。由于通过 HTTP 传送，所以在使用它们时要搭配 Spring Web 框架，也就是使用到 `DispatcherServlet`，可以改写 9.1.2 节的 HessianServerDemo 项目，只要修改一下 *service-config.xml* 就可以了：

HttpInvokerDemo　　service-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
       <property name="mappings">
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            <props>
                <prop key="/some.service">serviceExporter</prop>
            </props>
        </property>
    </bean>

    <bean id="someService"
          class="onlyfun.caterpillar.SomeServiceImpl"/>

    <bean id="serviceExporter"
          class="org.springframework.remoting.
                  → httpinvoker.HttpInvokerServiceExporter">
        <property name="service" ref="someService"/>
        <property name="serviceInterface"
              value="onlyfun.caterpillar.ISomeService"/>
    </bean>

</beans>
```

接下来客户端的部分，可以改写 HessianClientDemo 项目的内容，修改一下 Bean 定义文件的内容：

InvokerClientDemo　　invoker-client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someServiceProxy"
          class="org.springframework.remoting.
                    → httpinvoker.HttpInvokerProxyFactoryBean">
        <property name="serviceUrl" value="http://localhost:8080/
                    → HttpInvokerDemo/some.service"/>
        <property name="serviceInterface"
            value="onlyfun.caterpillar.ISomeService"/>
    </bean>

</beans>
```

注意到 "serviceUrl" 属性的设置，它是标准的 HTTP 请求地址，用来编写简单的客户端程序以使用 Http Invoker 服务器上的服务：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

InvokerClientDemo InvokerClient.java

```
package onlyfun.caterpillar;

import org.springframework.context.ApplicationContext;
import org.springframework.context.
           support.ClassPathXmlApplicationContext;

public class InvokerClient {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                  "invoker-client.xml");

        ISomeService service =
            (ISomeService) context.getBean("someServiceProxy");

        String result1 = service.doSomeService("Some request");
        System.out.println(result1);

        int result2 = service.doOtherService(1);
        System.out.println(result2);
    }
}
```

执行的结果与 RMIClientDemo 项目是相同的，可以参考图 9.1 的画面。

9.2 邮件

对于邮件发送服务的支持是由 Spring 的 `org.springframework.mail.MailSender` 接口所定义，它有两个实现类：`org.springframework.mail.cos.CosMailSenderImpl` 与 `org.springframework.mail.javamail.JavaMailSenderImpl`，前者支持 Jason Hunter's COS（在它的 Java Servlet Programming 书中提供的），后者支持 Java Mail。

9.2.1 简单邮件

在这里将以 Java Mail 的使用，来示范 Spring 如何提供 Java Mail 邮件服务的封装，首先来看如何实现最简单的纯文字邮件传送，程序的示范如下：

SimpleMailDemo SimpleMailDemo.java

```
package onlyfun.caterpillar;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.mail.javamail.JavaMailSenderImpl;
import org.springframework.mail.SimpleMailMessage;

public class SimpleMailDemo {
    public static void main(String[] args) throws Exception {
        JavaMailSenderImpl senderImpl = new JavaMailSenderImpl();
        // 设置 Mail Server
        senderImpl.setHost("your_mail_server.com");

        // 建立邮件消息
        SimpleMailMessage mailMessage = new SimpleMailMessage();

        // 设置收件人、寄件人、主题与正文
        mailMessage.setTo("xxx@your_mail_server.com");
        mailMessage.setFrom("xxx@your_mail_server.com");
        mailMessage.setSubject("Test");
        mailMessage.setText("This is a test!!!");

        // 传送邮件
        senderImpl.send(mailMessage);

        System.out.println("邮件传送 OK..");
    }
}
```

JavaMailSenderImple 默认会与端口 25 作沟通，如果您的 SMTP 邮件服务器不是使用 25 端口，可以使用 setPort() 来设置连接端口，执行以上程序时，记得加入相关 API 所需的 *.jar* 文件，在这里所使用的是 Spring 下载文件中 *lib* 目录下 *j2ee* 目录的 *activation.jar* 与 *mail.jar* 文件，传送出去的邮件在接收到后如图 9.2 所示。

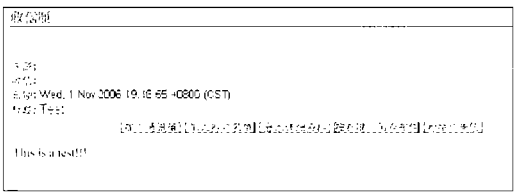


图 9.2　SimpleMailDemo 项目的执行结果

9.2.2　HTML 邮件

如果要使用 HTML 邮件，可以使用 Spring 的 org.springframework.mail.

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

javamail.MimeMessageHelper 来建立 HTML 邮件，直接使用实例来作示范：

HTMLMailDemo　　HTMLMailDemo.java

```
package onlyfun.caterpillar;

import org.springframework.mail.javamail.JavaMailSenderImpl;
import javax.mail.internet.MimeMessage;
import org.springframework.mail.javamail.MimeMessageHelper;

public class HTMLMailDemo {
    public static void main(String[] args) throws Exception {
        JavaMailSenderImpl senderImpl =
            new JavaMailSenderImpl();

        // 设置 Mail Server
        senderImpl.setHost("your_mail_server.com");

        // 建立邮件消息
        MimeMessage mailMessage =
            senderImpl.createMimeMessage();
        MimeMessageHelper messageHelper =
            new MimeMessageHelper(mailMessage);

        // 设置收件人、寄件人、主题与正文
        messageHelper.setTo("xxx@your_mail_server.com");
        messageHelper.setFrom("xxx@your_mail_server.com");
        messageHelper.setSubject("Test");
        messageHelper.setText(
                "<html><head></head><body><h1>Hello! Spring!"
                + "</h1></body></html>", true);

        // 传送邮件
        senderImpl.send(mailMessage);

        System.out.println("邮件传送 OK...");
    }
}
```

当 MimeMessageHelper 类的 setText() 方法上的 boolean 参数设置为 true 时，表示要启用 HTML 格式的邮件，来看一下传送出去后的邮件内容，如图 9.3 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

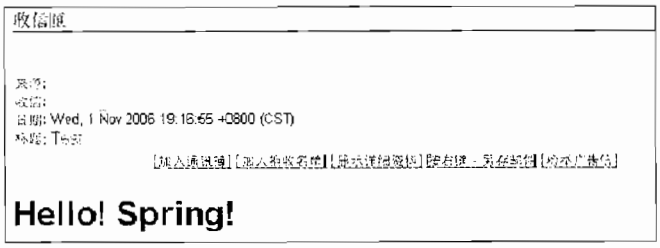


图 9.3　HTMLMailDemo 项目的执行结果

9.2.3　内嵌图片或附件

您也可以在邮件中内嵌图片，在内嵌图片时要给定一个 "cid" 值，直接来看程序的示范：

AttachedImageDemo　　AttachedImageDemo.java

```
package onlyfun.caterpillar;

import org.springframework.mail.javamail.JavaMailSenderImpl;
import javax.mail.internet.MimeMessage;
import org.springframework.core.io.ClassPathResource;
import org.springframework.mail.javamail.MimeMessageHelper;

public class AttachedImageDemo {
    public static void main(String[] args) throws Exception {
        JavaMailSenderImpl senderImpl =
                  new JavaMailSenderImpl();

        // 设置 Mail Server
        senderImpl.setHost("your_mail_server.com");

        // 建立邮件消息
        MimeMessage mailMessage =
            senderImpl.createMimeMessage();
        MimeMessageHelper messageHelper =
            new MimeMessageHelper(mailMessage, true);

        // 设置收件人、寄件人、主题与正文
        messageHelper.setTo("xxx@your_mail_server.com");
        messageHelper.setFrom("xxx@your_mail_server.com");
        messageHelper.setSubject("Test");
        messageHelper.setText(
            "<html><head></head><body><h1>Hello! Spring!"
            + "</h1><img src=\"cid:caterpillar\">"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            + "</body></html>", true);

        ClassPathResource img =
                new ClassPathResource("caterpillar.jpg");
        messageHelper.addInline("wish", img);

        // 传送邮件
        senderImpl.send(mailMessage);

        System.out.println("邮件传送成功...");
    }
}
```

在构造 MimeMessageHelper 类的实例时所给定的 boolean 值为 true，表示要启用 multipart 模式，在 setText() 方法指定 "cid" 值为 "caterpillar"，这个设置与 addInline() 方法中的 "cid" 值指定对应，addInline() 方法可以将影像文件加入至邮件中，来看一下传送后的邮件之画面（如图 9.4 所示）。



图 9.4 AttachedImageDemo 项目的执行结果

直接将影像文件加入邮件中有好有坏，好处是不用像给定连接的方式一样，必须在服务器上提供文件，才能让用户在读取邮件时取得所连接的图片，坏处是有的邮件客户端对于内嵌图片的支持会有问题，可能在某些客户端软件中会发生无法显示图片的情况。

您也可以在邮件中使用附加文件（Attachment file），只要使用 MimeMessageHelper 的 addAttachment() 方法，其中 "cid" 值是用来显示附加文件的名称，一个实例如下：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

AttachedFileDemo　　AttachedFileDemo.java

```
package onlyfun.caterpillar;

import org.springframework.mail.javamail.JavaMailSenderImpl;
import javax.mail.internet.MimeMessage;
import org.springframework.core.io.ClassPathResource;
import org.springframework.mail.javamail.MimeMessageHelper;

public class AttachedFileDemo {
    public static void main(String[] args) throws Exception {
        JavaMailSenderImpl senderImpl =
                    new JavaMailSenderImpl();

        // 设置 Mail Server
        senderImpl.setHost("your_mail_server.com");

        // 建立邮件消息
        MimeMessage mailMessage =
            senderImpl.createMimeMessage();
        MimeMessageHelper messageHelper =
            new MimeMessageHelper(mailMessage, true);

        // 设置收件人、寄件人、主题与正文
        messageHelper.setTo("xxx@your_mail_server.com");
        messageHelper.setFrom("xxx@your_mail_server.com");
        messageHelper.setSubject("Test");
        messageHelper.setText(
            "<html><head></head><body><h1>Hello! Spring!"
            + "</h1></body></html>", true);

        ClassPathResource file =
               new ClassPathResource("caterpillar.zip");
        messageHelper.addAttachment("caterpillar.zip", file);

        // 传送邮件
        senderImpl.send(mailMessage);

        System.out.println("邮件传送成功...");
    }
}
```

来看一下传送后的邮件画面（如图 9.5 所示）。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

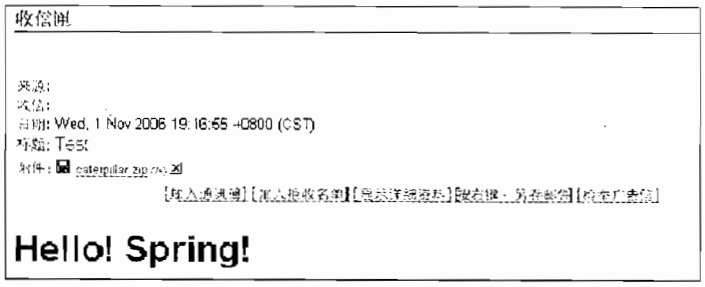


图 9.5 AttachedFileDemo 项目的执行结果

9.3 任务计划

对于任务计划定时执行的工作，JDK 的标准 API 提供了 java.util.Timer 与 java.util.TimerTask 类，Spring 则对它提供了抽象封装，让您可以善用 Spring 的容器管理功能，然而 Timer 功能有限，只能指定任务与任务之间的周期（Period），无法指定某个时间点定时执行任务，您可以使用 Quartz（*http://www.opensymphony.com/quartz/*），它提供了更多的任务计划功能，而 Spring 对 Quartz 进行了封装，在使用上更加方便。

9.3.1 使用 TimerTask

要定义一个计划任务（Task），可以继承 java.util.TimerTask 类，例如：

TimerTaskDemo — DemoTask.java

```
package onlyfun.caterpillar;

import java.util.TimerTask;

public class DemoTask extends TimerTask {
    public void run() {
        System.out.println("Task is executed.");
    }
}
```

接着可以使用 Spring 的 org.springframework.scheduling.timer.ScheduledTimerTask 来定义任务的执行周期，例如：

TimerTaskDemo — beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

      <bean id="demoTask" class="onlyfun.caterpillar.DemoTask"/>

      <bean id="scheduledTimerTask"
           class="org.springframework.scheduling.
                    → timer.ScheduledTimerTask">
         <property name="timerTask" ref="demoTask"/>
         <property name="period" value="600000"/>
         <property name="delay" value="10000"/>
      </bean>

      <bean id="timerFactoryBean"
           class="org.springframework.scheduling.
                    → timer.TimerFactoryBean">
         <property name="scheduledTimerTasks">
            <list>
               <ref bean="scheduledTimerTask"/>
            </list>
         </property>
      </bean>

    </beans>
```

在 ScheduledTimerTask 类的 "period" 属性中，定义的单位是毫秒，因此根据以上的定义，将每 10 分钟（600000 毫秒）执行一次所定义的任务，而 "delay" 属性定义了 Timer 启动后，在第一次执行任务前要延迟多少毫秒。

定义好的 ScheduledTimerTask 要使用 org.springframework.scheduling.timer.TimerFactoryBean 类来加入所有的计划任务，接下来只要 Spring 容器启动读取完定义文件，就会开始进行所排定的任务，例如：

TimerTaskDemo　　TimerTaskDemo.java

```
package onlyfun.caterpillar;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Timer;
import org.springframework.context.ApplicationContext;
import org.springframework.context.
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    support.ClassPathXmlApplicationContext;

public class TimerTaskDemo {
   public static void main(String[] args) throws IOException {
    ApplicationContext context =
          new ClassPathXmlApplicationContext("beans-config.xml");

       System.out.println("启动 Task..");
       System.out.println("请输入 exit 关闭 Task: ");

       BufferedReader reader =
          new BufferedReader(
                new InputStreamReader(System.in));

       while(true) {
          if(reader.readLine().equals("exit")) {
             break;
          }
       }

       Timer timer = (Timer) context.getBean("timerFactoryBean");
      timer.cancel();
   }
}
```

根据 Bean 定义文件的内容，这个程序在启动后 10 秒会执行第一次任务，之后每 10 分钟执行一次任务。

9.3.2 使用 MethodInvokingTimerTaskFactoryBean

使用 Spring 时，并不一定要继承 `TimerTask` 来定义一个任务，Spring 提供 `org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean`，可以直接指定执行某个对象的方法，例如可以改写一下 TimerTaskDemo 项目中的 `DemoTask` 类，这次不用继承 `TimerTask` 类：

TimerTaskDemo2 — DemoTask.java

```
package onlyfun.caterpillar;

public class DemoTask {
    public void execute() {
        System.out.println("Task is executed.");
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

接着只要在 Bean 定义文件中使用 MethodInvokingTimerTaskFactoryBean 即可，例如：

TimerTaskDemo2　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="demoTask" class="onlyfun.caterpillar.DemoTask"/>

    <bean id="timerTaskBean"
         class="org.springframework.scheduling.
              → timer.MethodInvokingTimerTaskFactoryBean">
       <property name="targetObject" ref="demoTask"/>
       <property name="targetMethod" value="execute"/>
    </bean>

    <bean id="scheduledTimerTask"
         class="org.springframework.scheduling.
              → timer.ScheduledTimerTask">
       <property name="timerTask" ref="timerTaskBean"/>
       <property name="period" value="5000"/>
       <property name="delay" value="1000"/>
    </bean>

    <bean id="timerFactoryBean"
         class="org.springframework.scheduling.
              → timer.TimerFactoryBean">
       <property name="scheduledTimerTasks">
          <list>
             <ref bean="scheduledTimerTask"/>
          </list>
       </property>
    </bean>

</beans>
```

执行时可以直接使用 TimerTaskDemo 项目中的 TimerTaskDemo 类，在底层，MethodInvokingTimerTaskFactoryBean 会自动建立 TimerTask 的实例以执行目标对象上的指定方法。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

9.3.3 使用 Quartz

JDK 标准 API 中所提供的 Timer 功能有限，只能指定任务与任务之间的周期（Period），无法指定某个时间点来定时执行任务，您可以使用 Quartz（*http://www.opensymphony.com/quartz/*），它提供了更多的任务计划功能，而 Spring 则对 Quartz 进行了封装，在使用上更加方便。

可以继承 `org.springframework.scheduling.quartz.QuartzJobBean` 来实现一个 Job 类，例如：

QuartzDemo — DemoJob.java

```
package onlyfun.caterpillar;

import org.quartz.JobExecutionContext;
import org.springframework.scheduling.
                        quartz.QuartzJobBean;

public class DemoJob extends QuartzJobBean {
    private JobData jobData;

    public void executeInternal(
                JobExecutionContext context) {
        System.out.println(
               jobData.getData() + " is executed.");
    }

    public void setJobData(JobData jobData) {
        this.jobData = jobData;
    }

    public JobData getJobData() {
        return jobData;
    }
}
```

JobData 只是一个 Job 数据对象的示范类，为了能看出计划 Job 被执行时的周期性，它传回一个 Date 对象，表示执行 Job 时所需数据的传回时间，例如：

QuartzDemo — JobData.java

```
package onlyfun.caterpillar;

import java.util.Date;

public class JobData {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public String getData() {
        return "Data from "
              + new Date().toString();
    }
}
```

直接来看定义文件如何定义：

QuartzDemo　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someData" class="onlyfun.caterpillar.JobData"/>

    <bean id="jobDetailBean"
         class="org.springframework.scheduling.
                    → quartz.JobDetailBean">
        <property name="jobClass"
            value="onlyfun.caterpillar.DemoJob"/>
        <property name="jobDataAsMap">
            <map>
                <entry key="jobData" value-ref="someData"/>
            </map>
        </property>
    </bean>

    <bean id="simpleTriggerBean"
         class="org.springframework.scheduling.
                    → quartz.SimpleTriggerBean">
        <property name="jobDetail" ref="jobDetailBean"/>
        <property name="repeatInterval" value="5000"/>
        <property name="startDelay" value="1000"/>
    </bean>

    <bean id="schedulerFactoryBean"
         class="org.springframework.scheduling.
                    → quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="simpleTriggerBean"/>
            </list>
        </property>
    </bean>

</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在以上设置中特别要注意的是，org.springframework.scheduling.quartz.JobDetailBean 的 "jobClass" 属性必须提供 Job 的类名称，而不是 Job 的 Bean 实例，Job 所需的数据可以在 "jobDataAsMap" 属性中来提供。

在计划任务的周期指定上，使用 org.springframework.scheduling.quartz.SimpleTriggerBean 来指定，这点与 TimerTask 任务计划中的指定方式类似，指定的时间同样也是以毫秒作为单位，而排定 Job 时，所使用的是 org.springframework.scheduling.quartz.SchedulerFactoryBean。

完成设置之后，只要在启动 Spring 并读取了定义文件后，计划任务就会进行，例如编写一个简单的任务启动类：

QuartzDemo　　QuartzDemo.java

```
package onlyfun.caterpillar;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import org.quartz.Scheduler;
import org.quartz.SchedulerException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.
            support.ClassPathXmlApplicationContext;

public class QuartzDemo {
    public static void main(String[] args)
                throws IOException, SchedulerException {
     ApplicationContext context =
            new ClassPathXmlApplicationContext("beans-config.xml");
        System.out.println("启动 Task..");
        System.out.println("请输入 exit 关闭 Task: ");

        BufferedReader reader =
            new BufferedReader(
                    new InputStreamReader(System.in));

        while(true) {
            if(reader.readLine().equals("exit")) {
                break;
            }
        }

        Scheduler scheduler =
          (Scheduler) context.getBean("schedulerFactoryBean");
        scheduler.shutdown();
    }
}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

来看看这个程序执行时的一个参考画面（如图 9.6 所示）。当中可以看到每个任务的执行时间：

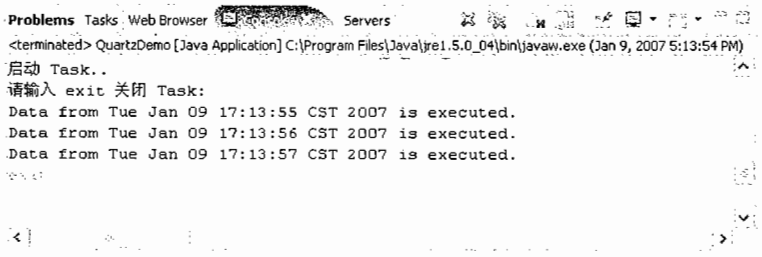


图 9.6　QuartzDemo 项目的执行结果

使用 SimpleTriggerBean 只能作简单的 Job 与 Job 之间执行的周期（Period）指定，如果要直接作时间点的指定，则可以使用 org.springframework.scheduling.quartz.CronTriggerBean，例如：

QuartzDemo2　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someData" class="onlyfun.caterpillar.JobData"/>

    <bean id="jobDetailBean"
         class="org.springframework.scheduling.
                 → quartz.JobDetailBean">
        <property name="jobClass"
            value="onlyfun.caterpillar.DemoJob"/>
        <property name="jobDataAsMap">
            <map>
                <entry key="jobData" value-ref="someData"/>
            </map>
        </property>
    </bean>

    <bean id="cronTriggerBean"
         class="org.springframework.scheduling.
                 → quartz.CronTriggerBean">
        <property name="jobDetail" ref="jobDetailBean"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <property name="cronExpression" value="0 0 19 * * ?"/>
    </bean>

    <bean id="schedulerFactoryBean"
          class="org.springframework.scheduling.
                  → quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="cronTriggerBean"/>
            </list>
        </property>
    </bean>

</beans>
```

重点在于 "cronExpression" 属性的指定，指定的格式是至少 6 个时间元素，最多 7 个时间元素，例如上面的指定是每天的 19 时要执行 Job 一次，"cronExpression" 属性指定的格式如下：

- 秒（0-59）
- 分（0-59）
- 小时（0-23）
- 每月第几天（1-31）
- 月（1-12 或 JAN-DEC）
- 每星期第几天（1-7 或 SUN-SAT）
- 年（1970-2099）

其中“每月第几天”与“每星期第几天”是互斥的，两个只能设置一个，不设置的以“?”符号编写，如果有好几个时间点，可以使用“,” 符号，例如：“0 0 10,12,14 * * ?”表示每天的 10 时、12 时、14 时要执行 Job；对于连续的时间可以使用“ - ”符号，例如“0 0 10,12,14 1-15 * ?”表示每月的 1 到 15 日每 10 时、12 时、14 时要执行 Job，时间格式中的年指定可有可无，例如：“0 0 10,12,14 ? * MON 2006”表示 2006 年每星期一的 10 时、12 时、14 时要执行 Job。

9.3.4 使用 MethodInvokingJobDetailFactoryBean

如果使用 Spring，并不一定要继承 QuartzJobBean 类来定义一个 Job 类，Spring 提供 org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean，可

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

以直接指定执行某个对象的方法，例如改写一下 QuartzDemo 项目中的 DemoJob 类：

QuartzDemo3　　DemoJob.java

```
package onlyfun.caterpillar;

public class DemoJob {
    private JobData jobData;

    public void execute() {
        System.out.println(
                jobData.getData() + " is executed.");
    }

    public void setJobData(JobData jobData) {
        this.jobData = jobData;
    }

    public JobData getJobData() {
        return jobData;
    }
}
```

接着只要在 Bean 定义文件中使用 MethodInvokingJobDetailFactoryBean 即可，例如改写一下 QuartzDemo 项目中的 *beans-config.xml*：

QuartzDemo3　　beans-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="someData" class="onlyfun.caterpillar.JobData"/>

    <bean id="someJob"
          class="onlyfun.caterpillar.DemoJob">
        <property name="jobData" ref="someData"/>
    </bean>

    <bean id="jobDetailBean"
          class="org.springframework.scheduling.
                  → quartz.MethodInvokingJobDetailFactoryBean">
        <property name="targetObject" ref="someJob"/>
        <property name="targetMethod" value="execute"/>
    </bean>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <bean id="cronTriggerBean"
         class="org.springframework.scheduling.
                → quartz.CronTriggerBean">
        <property name="jobDetail" ref="jobDetailBean"/>
        <property name="cronExpression" value="0 30 18 * * ?"/>
    </bean>

    <bean id="schedulerFactoryBean"
         class="org.springframework.scheduling.
                → quartz.SchedulerFactoryBean">
        <property name="triggers">
            <list>
                <ref bean="cronTriggerBean"/>
            </list>
        </property>
    </bean>

</beans>
```

在上面的指定中，每天的 18 时 30 分都会执行指定的任务，在底层，MethodInvokingJobDetailFactoryBean 会自动建立 JobDetail 的实例，以执行目标对象的指定方法。

9.4 接下来的主题

到这个章节为止，本书对于 Spring 个别功能的介绍先告一段落，事实上关于 Spring 的功能还有许多，因篇幅与个人所涉猎的范围有限，无法每一个都详加介绍，建议您可以参考 Spring 所提供的参考手册，有了本书的基础加上参考手册的对照，相信对于 Spring 的其他功能都可以很快上手。



图 9.7 体验 Spring 的开发快感

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

第

项目：Spring 在线书签

章

在先前的各章节中，着重于介绍 Spring 的各个功能，并在一些简单的应用程序中，突显 Spring 中各种 API 的使用与配置的设置方式。

在这个章节中，将组合先前介绍过的 Spring 功能，完成一个可以在线管理书签的 Web 程序，程序中将使用 Spring Web MVC 框架、Spring IoC 容器、Hibernate 以及 Java Mail 等，在 View 层技术上则使用 JSP、Spring 标签与 JSTL，借以实际完成 Spring 在线书签，您可以了解到如何使用 Spring 来开发项目。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

10.1　程序概观

在正式开始设计程序之前，先花一些时间来设计程序的架构与考虑所要采用的技术，虽然这只是一个简单的 Web 程序，但事先做一些简单的纸上作业，可以避免在程序开发过程中，因架构不清楚而造成程序混乱或不必要的修改。

10.1.1　架构规划

Spring 在线书签将采用 Web MVC 的架构来设计程序，基于 Web MVC 架构的流程，规划出如图 10.1 所示的流程架构：

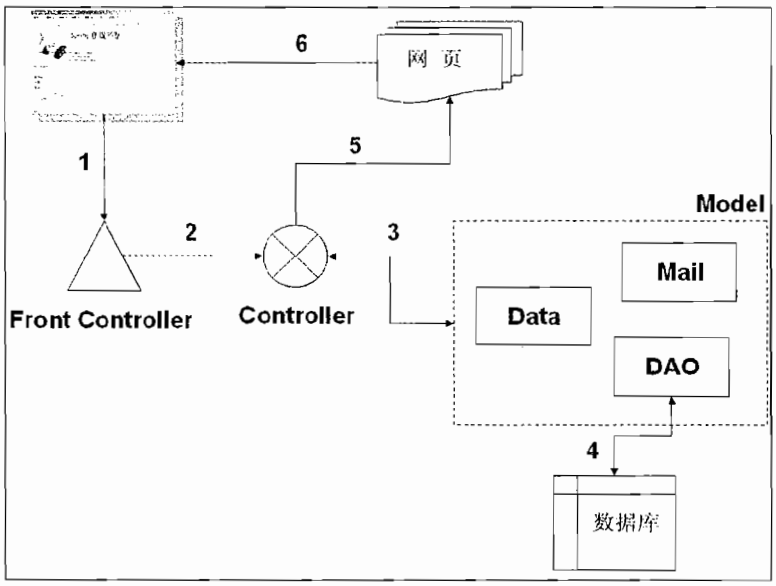


图 10.1　Spring 在线书签流程架构

对于图中每个流程与图标的作用，稍微作个说明：

- 前端控制器（Front Controller）

在 Spring 在线书签中，采用前端控制器来接受用户的请求，用户在“流程 1”中向前端控制器发送请求，前端控制器将决定由哪一个控制器来处理请求。

- 控制器

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

前端控制器决定好控制器后，在“流程 2”中将请求交由控制器来处理，控制器将撷取用户的请求参数、进行数据验证，并呼叫业务层的对象来完成请求。

- Data、Mail、DAO 等 Model 对象

在请求中有一些业务必须调用业务层中的 Model 对象来处理，控制器在“流程 3”中调用数据存取对象（DAO）、Mail 业务对象等来处理这些请求，如果需要的话，还可以将一些消息封装为数据（Data）对象。

- 数据库

Spring 在线书签程序中用来储存书签的媒介将采用数据库，在“流程 4”中数据存取对象将负责与数据库进行储存、查询、更新等动作。

- 网页

在处理流程返回至控制器之后，控制器收集必要的消息并设置为网页可以存取的数据对象，接着在“流程 5”中转发给呈现结果画面的网页，网页会撷取必要的数据对象并显示画面，在“流程 6”中将结果传回给用户。

10.1.2 采用技术

在了解了大致的程序架构之后，来考虑每一个环节将实际采用的技术。

- 前端控制器（Front Controller）

在 Spring 在线书签中，将使用 Spring 的 Web MVC 框架来实现 Web MVC 架构，其中所采用的前端控制器为 Spring 的 DispatcherServlet 类。

- 控制器

由于采用 Spring Web MVC 框架，配合 Spring 的 DispatcherServlet 类，将使用 Spring Web MVC 中 Controller 继承架构上的类，例如 AbstractCommandController、SimpleFormController、MultiActionController 等类。

- Data、Mail、DAO 等 Model 对象

数据对象为自制的 User 类与 Bookmark 类，在 DAO 对象的实现上，将采用 Hibernate 来自动进行对象/关系映射（Object/Relational Mapping），在数据存取的过程中，将会使用到 User、Bookmark 类作为数据对象，而程序中会有发送邮件的需求，这个部分使用到 Java Mail 的功能，会使用 Spring 的 Java Mail 抽象层封装。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

- **数据库**

Spring 在线书签用来储存用户数据与书签的媒介，将采用 MySQL 关系型数据库。

- **网页**

在网页技术的采用上，使用 JSP 技术为基础，运用 Spring 的一些绑定标签，并使用 JSTL 来处理页面逻辑的呈现。

10.1.3　功能概述

Spring 在线书签，可以让您进行在线书签管理，您可以注册一个新的账号，并拥有自己的书签管理页面，可以在管理页面中进行书签的新增与删除动作，或者是变更您的密码，如果忘记密码的话，还可以输入注册时的邮件地址，让系统发一封信件告知您的用户密码。

在这里先不列出程序执行的画面，程序执行的画面可以在稍后的章节中设计相关页面时看到。

良葛格的话匣子<<<

基于篇幅的限制，这个程序示范的，仅是如何组合相关的框架或 API 来完成一个程序，程序中的一些功能并没有设计得面面俱到，这个程序主要目的是教学之用，如果您打算实际在某些场合使用这个程序，必须自行修改、完备这个程序。

10.2　Model 设计

在 Spring 在线书签的设计中，业务层的 Model 对象负责与数据库进行存取的 UserDAO、BookmarkDAO，它们分别负责存取用户数据与书签数据，而在用户忘记密码时，必须使用到邮件功能寄送用户的密码，这将设计一个 SimpleMail 类来完成这个功能。

10.2.1　UserDAO

对于用户数据的储存，程序中设计了一个 User 类，用以对应数据库表格中的 user 表格，来看一下 User 类的编写：

SpringBookmark　　User.java

```
package onlyfun.caterpillar.model;

public class User {
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    private Integer id;
    private String email;
    private String username;
    private String passwd;

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getId() {
        return id;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }

    public String getEmail() {
        return email;
    }

    public String getUsername() {
        return username;
    }

    public String getPasswd() {
        return passwd;
    }
}
```

可以看到 User 对象中将包括用户的名称、密码与邮件地址，其中 "id" 属性是在运用 Hibernate 时所需的唯一识别，这个 User 类将对应于数据库中的 user 表格，可以使用以下的 SQL 来建立 user 表格：

```
CREATE TABLE user  (
  id INT(11) NOT NULL auto_increment PRIMARY KEY,
  username VARCHAR(16),
  passwd CHAR(16) NOT NULL,
  email VARCHAR(100) NOT NULL
);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

在 UserDAO 的实现上，将使用 Hibernate 进行对象关系映射，为了映射 User 类与 user 表格，需要编写一个 *User.hbm.xml*，如下所示：

SpringBookmark　　User.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="onlyfun.caterpillar.model.User"
           table="user">
        <id name="id" column="id">
            <generator class="native"/>
        </id>
        <property name="username" column="username"/>
        <property name="passwd" column="passwd"/>
        <property name="email" column="email"/>
    </class>
</hibernate-mapping>
```

为了不让应用程序依赖于实际的 DAO 实现类，可以通过设计一个 IUserDAO 接口，让应用程序依赖于接口，接口设计如下：

SpringBookmark　　IUserDAO.java

```
package onlyfun.caterpillar.model;

public interface IUserDAO {
    public void insert(User user);
    public User findByName(String username);
    public User findByEmail(String email);
    public void update(User user);
}
```

在 IUserDAO 中，可以根据 User 对象来进行数据的储存与更新，并也可以分别根据用户的名称，或是用户的邮件地址来查询 User 数据，在 IUserDAO 接口的实现类 UserDAO 的设计上，如下所示：

SpringBookmark　　UserDAO.java

```
package onlyfun.caterpillar.model;

import java.util.List;
import org.springframework.orm.hibernate3.HibernateTemplate;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
public class UserDAO implements IUserDAO {
    private HibernateTemplate hibernateTemplate;

    public void setHibernateTemplate(
           HibernateTemplate hibernateTemplate) {
        this.hibernateTemplate = hibernateTemplate;
    }

    public void insert(User user) {
        hibernateTemplate.saveOrUpdate(user);
    }

    public User findByName(String username) {
       if(username.equals("")) {
         return null;
       }
        List users = hibernateTemplate.find(
               "from User user where user.username='"
               + username + "'");
        return (User) users.get(0);
    }

    public User findByEmail(String email) {
        List users = hibernateTemplate.find(
               "from User user where user.email='"
               + email + "'");
        return (User) users.get(0);
    }

    public void update(User user) {
        hibernateTemplate.update(user);
    }
}
```

良葛格的话匣子<<<

在 HibernateTemplate 的 find()方法中看到的是 Hibernate Query Language，简称 HQL，为 Hibernate 所鼓励使用的查询语言，在使用上比 SQL 简单且具有对象导向语言的特性，可以参考我的网站中有关 HQL 的介绍：

http://caterpillar.onlyfun.net/Gossip/HibernateGossip/ HibernateGossip.html

在 Spring 在线书签中，使用 Spring 对 Hibernate 的封装来运用 Hibernate（参考第 6 章的内容），这里使用 HibernateTemplate 对象来进行各种数据存取、查询、更新操作，HibernateTemplate 对象将使用 Spring 的依赖注入功能来注入实例，稍后可以看到配置文件

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

中会进行相关的设置。

10.2.2　BookmarkDAO

对于书签数据的储存，程序中设计了一个 Bookmark 类，用以对应数据库表格中的 bookmark 表格，来看一下 Bookmark 类的编写：

SpringBookmark　　Bookmark.java

```
package onlyfun.caterpillar.model;

public class Bookmark {
    private Integer id;
    private String username;
    private String url;

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getId() {
        return id;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUsername() {
        return username;
    }
}
```

可以看到 Bookmark 对象中将包括书签的拥有者名称与书签网址，其中 "id" 属性是在运用 Hibernate 时所需的唯一识别，这个 Bookmark 类将对应于数据库中的 bookmark 表格，

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

可以使用以下的 SQL 来建立 bookmark 表格：

```
CREATE TABLE bookmark (
  id INT(11) NOT NULL auto_increment PRIMARY KEY,
  username VARCHAR(16) NOT NULL,
  bm_URL VARCHAR(255) NOT NULL
);
```

良葛格的话匣子<<<

实际上，user 表格与 bookmark 表格可以建立为一对多关系，不过由于这本书中并没有介绍到 Hibernate 如何映射一对多关系，因而范例仍设计为两个独立的表格。

在 BookmarkDAO 的实现上，将使用 Hibernate 进行对象关系映射，为了映射 Bookmark 类与 bookmark 表格，需要编写一个 *Bookmark.hbm.xml*，如下所示：

SpringBookmark　　Bookmark.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="onlyfun.caterpillar.model.Bookmark"
           table="bookmark">
        <id name="id" column="id">
            <generator class="native"/>
        </id>
        <property name="username" column="username"/>
        <property name="url" column="bm_URL"/>
    </class>
</hibernate-mapping>
```

为了不让应用程序依赖于实际的 DAO 实现类，可以设计一个 IBookmarkDAO 接口，让应用程序依赖于接口，接口设计如下：

SpringBookmark　　IBookmarkDAO.java

```
package onlyfun.caterpillar.model;

import java.util.List;

public interface IBookmarkDAO {
    public List findUserUrl(String username);
    public void addUserUrl(String username, String url);
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void deleteUserUrls(String username, String[] urls);
}
```

在 IBookmarkDAO 中，规范的方法有：根据用户名称与提供的书签 URL 来增加书签数据，根据用户名称来查询书签数据，进行书签的删除。在 IBookmarkDAO 接口的实现类 BookmarkDAO 的设计上，如下所示：

SpringBookmark　　BookmarkDAO.java

```
package onlyfun.caterpillar.model;

import java.util.List;
import org.springframework.orm.hibernate3.HibernateTemplate;

public class BookmarkDAO implements IBookmarkDAO {
    private HibernateTemplate hibernateTemplate;

    public void setHibernateTemplate(
            HibernateTemplate hibernateTemplate) {
        this.hibernateTemplate = hibernateTemplate;
    }

    public List findUserUrl(String username) {
        List list= hibernateTemplate.find(
          "select bm.url from Bookmark as bm where bm.username='"
                + username + "'");
        return list;
    }

    public void addUserUrl(String username, String url) {
        Bookmark bm = new Bookmark();
        bm.setUsername(username);
        bm.setUrl(url);
        hibernateTemplate.saveOrUpdate(bm);
    }

    public void deleteUserUrls(
            String username, String[] urls) {
        for(int i = 0; i < urls.length; i++) {
            List list = hibernateTemplate.find(
                    "from Bookmark bm where (bm.username='"
                    + username +
                    "') and (bm.url='" + urls[i]+ "')");
            hibernateTemplate.delete(list.get(0));
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        }
    }
}
```

同样的在 BookmarkDAO 的实现中，使用 Spring 对 Hibernate 的封装来运用 Hibernate，HibernateTemplate 对象也将使用 Spring 的依赖注入功能来注入实例，稍后可以看到配置文件中会进行相关的设置。

10.2.3 SimpleMail

当用户忘记登录时所使用的密码时，可以要求他输入注册时所使用的 E-mail 地址，以查询用户的密码，程序会寄送邮件告知密码，邮件的寄送在程序中使用 Java Mail 来完成，并运用 Spring 对 Java Mail 的封装，以简化程序的编写（参考 9.2.1 节），这个部分比较简单，来看一下 SimpleMail 类的设计：

SpringBookmark　　SimpleMail.java

```
package onlyfun.caterpillar.model;

import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSenderImpl;

public class SimpleMail {
    private String smtpHost;
    private String from;

    public void sendPasswdMail(String email,
            String username, String passwd) {
        JavaMailSenderImpl senderImpl = new JavaMailSenderImpl();
        senderImpl.setHost(getSmtpHost());

        SimpleMailMessage mailMessage = new SimpleMailMessage();

        mailMessage.setTo(email);
        mailMessage.setFrom(getFrom());
        mailMessage.setSubject("您的密码");
        mailMessage.setText(username + " 您好，您的密码是 " + passwd);

        senderImpl.send(mailMessage);
    }

    public void setSmtpHost(String smtpHost) {
        this.smtpHost = smtpHost;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    public String getSmtpHost() {
        return smtpHost;
    }

    public void setFrom(String from) {
        this.from = from;
    }

    public String getFrom() {
        return from;
    }
}
```

其中有关于 SMTP 服务器与寄件人邮件地址，可以使用 Spring 来进行依赖注入，而不用直接编写在程序之中，如此在更换这些设置时就很方便。

10.3　View 设计

在 Spring 在线书签中，将使用 JSP 作为网页基础，使用 Tag Files 来设计可重复使用的网页组件，并运用 Spring 绑定标签与 JSTL 来进行页面逻辑的呈现。

10.3.1　Tag Files

网页中有许多共同的部分，像是网页开头（Header）、结尾（Footer）或是网站信息、用户功能选单等，这些可以使用 JSP 2.0 技术中的 Tag Files 设计为 *.tag* 文件，并在必要的时候包括在网页之中，如此在编写网页时就不用重复编写相同的内容了。

网页开头的 Tag File 如下编写，包括了基本的样式、网站名称与 Logo 的设置：

SpringBookmark　　header.tag

```
<%@tag description="每个网页的 Header" pageEncoding="BIG5"%>
<html>
<head>
    <title>Spring 在线书签</title>
    <style>
    body {font-family: Arial, Helvetica, sans-serif;
          font-size: 13px}
    li, td {Arial, Helvetica, sans-serif;
            font-size: 13px}
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    hr {color: #3333cc; width: 300;
       text-align: left}
    a {color: #000000}
    </style>
</head>
<body>
<img src='images/caterpillar.jpg'
     alt='Spring 在线书签 Logo' border='0' align='left'
     valign='bottom'/>
<h1> Spring 在线书签</h1>
<hr style="width: 100%; height: 2px;">
<hr style="width: 100%; height: 2px;">
```

网站中有些部分要出现网址的相关消息，这可以编写一个 *site_info.tag* 文件：

SpringBookmark site_info.tag

```
<%@tag description="网站消息" pageEncoding="BIG5"%>
<ul>
  <li>在线储存您的书签
  <li>分享您我的书签
  <li>快来秀出最酷的链接
</ul>
```

在用户登录网页之后，必须出现功能选单供用户进行功能选择，每一个画面中都要出现，因此也可以将功能选单设计为可重复使用的 Tag File，例如：

SpringBookmark usermenu.tag

```
<%@tag description="用户功能表单" pageEncoding="BIG5"%>
<hr style="width: 100%; height: 2px;">
<a href='member.do'>会员首页</a>  | 
<a href='member.do?action=addUrl'>加入书签</a>  | 
<a href='member.do?action=changePasswd'>
改变密码</a>  | 
<a href='member.do?action=logout'> </a>
<hr style="width: 100%; height: 2px;">
```

最后别忘了，由于每个网页的结尾都是差不多的，因此可以设计一个 *footer.tag* 文件：

SpringBookmark footer.tag

```
<%@tag description="每个网页的 Footer" pageEncoding="BIG5"%>
</body>
</html>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

设计完以上可重复使用的 Tag Files 后，以后在编写网页时，如果在开头、结尾或是网址消息等需要更改时，可直接更改 Tag Files，以便于在下一次请求时使用到这些 Tag Files 的网页，也就会自动更新内容。

10.3.2　注册网页

当用户第一次来到网站时，必须先注册一个账号以成为合法的用户，所以必须给用户提供一个注册网页，这里可以使用 Spring 的绑定标签（参考 8.1.2 节），它将接受用户输入的 Command 对象绑定至网页上的字段，以提供相对应于字段的消息，注册网页的设计如下所示：

SpringBookmark　　register.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
   pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>
<%@taglib prefix="spring"
              uri="http://www.springframework.org/tags"%>

<caterpillar:header/>
   <spring:bind path="command.*">
      <font color="red">
         <b>${status.errorMessage}</b>
      </font><br>
   </spring:bind>
 <form method='post' action='register.do'>
 <table bgcolor=#cccccc>
   <tr>
    <td>邮件地址: </td>
    <td>
       <spring:bind path='command.email'>
       <input type='text' name='${status.expression}'
                value='${status.value}' size='30'
                maxlength='100'>
                </spring:bind>
    </td>
   </tr>
   <tr>
    <td>名称（最大 16 字符）: </td>
    <td valign='top'>
       <spring:bind path='command.username'>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <input type='text' name='${status.expression}'
                        value='${status.value}'
                        size='16' maxlength='16'>
        </spring:bind>
      </td>
    </tr>
    <tr>
      <td>密码（6 到 16 字符）：</td>
      <td valign='top'>
        <spring:bind path='command.passwd'>
        <input type='password' name='${status.expression}'
                  size='16' maxlength='16'>
        </spring:bind>
      </td>
    </tr>
    <tr>
      <td>确认密码：</td>
      <td>
        <spring:bind path='command.passwd2'>
        <input type='password' name='${status.expression}'
              size='16' maxlength='16'>
        </spring:bind>
      </td>
    </tr>
    <tr>
      <td colspan='2' align='center'>
      <input type='submit' value='注册'></td></tr>
  </table></form>
<caterpillar:footer/>
```

执行网页后，所呈现的画面如图 10.2 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

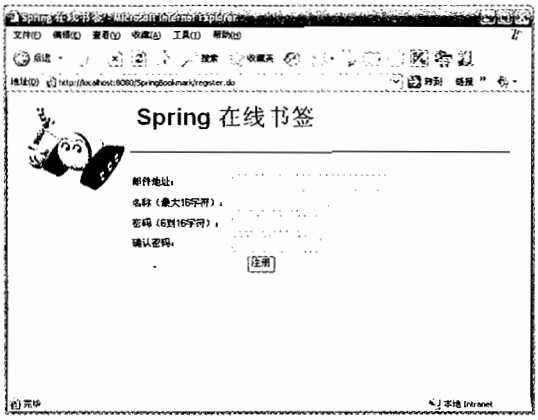


图 10.2　注册网页

注册成功，就会出现一个注册成功的网页，并直接将用户登录，用户可以直接发现到一个连接会员网页的链接，注册成功的网页如下所示：

SpringBookmark　　register_success.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>
<caterpillar:header/>
您的注册已经成功，前往会员网页！
<br><a href='member.do'>会员网页</a><br>
<caterpillar:footer/>
```

执行这个 JSP 网页后的画面如图 10.3 所示。



图 10.3　注册成功网页

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

10.3.3 登录网页

如果用户已经注册了，当连接到应用程序首页时，可以显示一个登录网页，用户提供输入名称与密码进行登录，网页的编写如下：

SpringBookmark　　login.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
   pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>
<%@taglib prefix="spring"
              uri="http://www.springframework.org/tags"%>

<caterpillar:header/>
<caterpillar:site_info/>
  <a href='register.do'>还不是会员？</a><p>
  <form method='POST' action='login.do'>
    <spring:bind path="command.*">
       <font color="red">
          <b>${status.errorMessage}</b>
       </font><br>
    </spring:bind>
  <table bgcolor='#cccccc'>
   <tr>
     <td colspan='2'>会员登录</td>
   <tr>
     <td>名称: </td>
     <td>
         <spring:bind path='command.username'>
         <input type='text' name='${status.expression}'
              value='${status.value}'>
         </spring:bind>
     </td>
   </tr>
   <tr>
     <td>密码: </td>
     <td>
        <spring:bind path='command.passwd'>
        <input type='password'
              name='${status.expression}'>
        </spring:bind>
     </td>
   </tr>
   <tr>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    <td colspan='2' align='center'>
    <input type='submit' value='登录'></td></tr>
  <tr>
    <td colspan='2'><a href='forgot.do'>忘记密码? </a></td>
  </tr>
 </table></form>
<caterpillar:footer/>
```

网页执行之后的画面如图 10.4 所示。

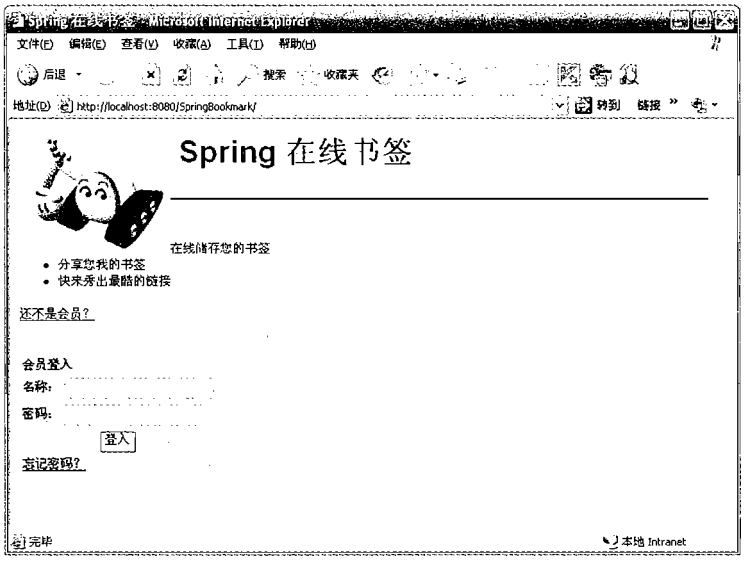


图 10.4 登录网页

登录成功的话，会出现一个登录成功的网页，用户可以直接连接到会员网页，登录成功的网页如下所示：

SpringBookmark　　login_success.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>
<caterpillar:header/>
登录成功，前往会员网页!
<br><a href='member.do'>会员网页</a><br>
<caterpillar:footer/>
```

一个登录成功的画面如图 10.5 所示。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

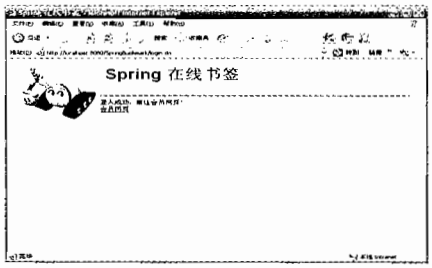


图 10.5 登录成功网页

10.3.4 会员网页

当用户登录并连接至会员网页之后，默认会显示用户目前所拥有的书签，在书签网页上可以让用户选择是否删除某些书签，并提供有其他功能的选单，让用户进行不同的功能操作，会员网页的设计如下所示：

SpringBookmark member.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<caterpillar:header/>

  <br> ${valid_user} 您好: <br><br>
  <form name='bm_table' action='member.do' method='post'>
  <table width='300' cellpadding='2' cellspacing='0'>
  <tr bgcolor='#cccccc'><td><strong>书签</strong></td>
  <td><strong>删除? </strong></td></tr>
  <c:forEach var="url" items="${urls}" >
     <tr bgcolor='#ffffff'>
        <td><a href=<c:out value="${url}"/>>
              <c:out value="${url}"/>
           </a></td>
        <td><input type='checkbox' name='del_me'
                value=<c:out value="${url}"/>></td>
     </tr>
  </c:forEach>
       <tr bgcolor='#ffffff'>
         <td></td>
         <td>
           <input type='hidden' name='action'
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                value='deleteUrl'>
            <input type='submit' value='删除'>
          </td>
        </tr>
      </table>
      </form>

<caterpillar:usermenu/>
<caterpillar:footer/>
```

由于事先无法决定用户所拥有的书签数量，为了安排页面逻辑，页面中使用了 JSTL 的 <forEach> 标签以及 <out> 标签来重复输出书签链接的 HTML，如此在网页中就不用编写 Scriplet 了，来看一个有书签时的参考画面（如图 10.6 所示）。

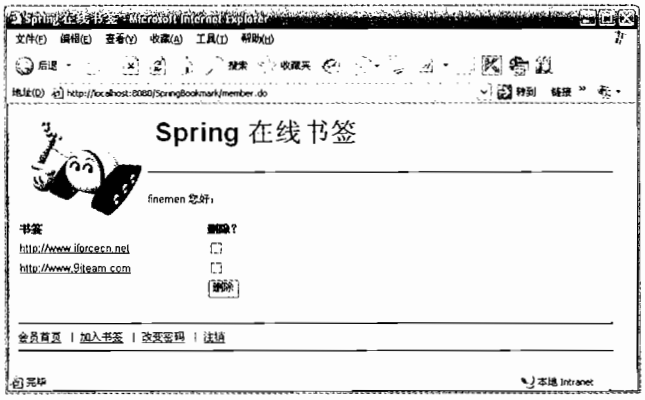


图 10.6　会员网页

10.3.5　加入书签网页

当点击会员网页下方功能表中的“加入书签”时，可以连接至加入书签的网页，用户在这个网页中输入网站的 URL，以进行书签的新增，加入书签的网页编写如下：

SpringBookmark　　addbm.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>
<caterpillar:header/>

<form name='bm_table' action='member.do' method='post'>
<table width='300' cellpadding='2'
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
      cellspacing='0' bgcolor='#cccccc'>
<tr><td>新书签: </td>
    <td><input type='text'
               name='new_url'  value='http://'
               size='30' maxlength='255'>
    </td>
</tr>
<tr><td colspan='2' align='center'>
<input type='hidden' name='action' value='addUrl'>
<input type='submit' value='新增书签'></td></tr>
</table>
</form>

<caterpillar:usermenu/>
<caterpillar:footer/>
```

执行的画面如图 10.7 所示。

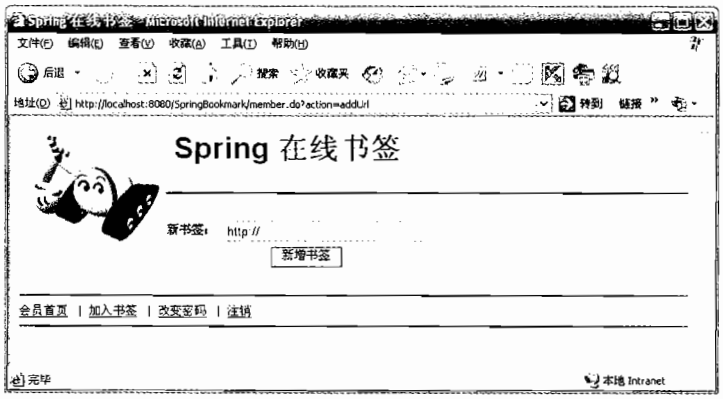


图 10.7 加入书签网页

10.3.6 改变、取回密码网页

用户在登录之后，可以点击“改变密码”链接，然后直接进行密码的更改，网页编写如下：

SpringBookmark change_passwd.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>
<caterpillar:header/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
<br>
<form action='member.do' method='post'>
<table width='250' cellpadding='2'
      cellspacing='0' bgcolor='#cccccc'>
<tr><td>新密码: </td>
    <td>
      <input type='password' name='new_passwd'
            size='16' maxlength='16'>
      <input type='hidden' name='action'
            value='changePasswd'>
    </td>
</tr>
<tr><td colspan='2' align='center'>
       <input type='submit' value='改变密码'>
</td></tr>
</table>
<br>
</form>

<caterpillar:usermenu/>
<caterpillar:footer/>
```

改变密码的网页呈现如图 10.8 所示。

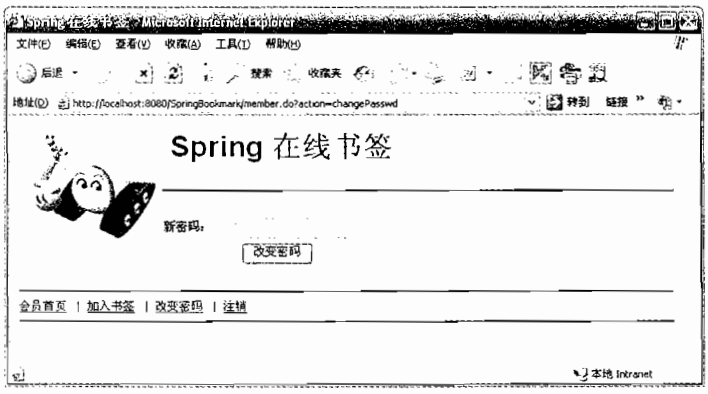


图 10.8　改变密码网页

如果用户因为忘记密码无法登录，则可以在登录网页的下方选择“忘记密码？”链接，然后输入注册时所使用的 E-mail 地址，系统就可以发一封邮件给用户告知密码，该网页的设计如下所示：

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

SpringBookmark forgot.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<%@taglib prefix="caterpillar" tagdir="/WEB-INF/tags" %>

<caterpillar:header/>
   <br>
   <form action='forgot.do' method='post'>
   <table width='250' cellpadding='2'
         cellspacing='0' bgcolor='#cccccc'>
   <tr><td>输入邮件地址: </td>
      <td><input type='text' name='email'
               size='16' maxlength='100'></td>
   </tr>
   <tr><td colspan='2' align='center'>
         <input type='submit' value="寄出密码">
   </td></tr>
   </table>
   <br>
<caterpillar:footer/>
```

网页执行时的画面如图 10.9 所示。



图 10.9 取回密码网页

10.3.7 首页重新导向

在 Spring 在线书签中，所有的用户请求都要先经过前端控制器，如果用户直接使用 *http://localhost:8080/SpringBookmark/*这样的网址，会连接至服务器默认的 Welcome 网页，您应该提供一个 *index.jsp* 作为 Welcome 网页，当用户连接至这个网页时，将重新导向至

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

http://localhost:8080/SpringBookmark/login.do，以进行用户登录，*index.jsp* 的设计如下所示：

SpringBookmark　　index.jsp

```
<%@ page language="java" contentType="text/html; charset=BIG5"
    pageEncoding="BIG5"%>
<jsp:forward page="login.do"/>
```

到这里为止，所有的网页设计完成，可以看到，网页中没有任何的 Java 程序代码出现，有的只是网页标签，这是使用 Web MVC 架构的好处，程序逻辑与画面呈现逻辑不会混杂在一起。

10.4　Controller 设计

前一节中所设计的 JSP 网页必须搭配相关的 Controller、Command 或是数据对象，才可以呈现出所需的数据，Controller 对象也是将 Model 与 View 结合在一起的主要关键。本节来看 Spring 在线书签中 Controller 的相关设计。

10.4.1　RegisterController

在用户进行注册的时候，可以使用一个 RegisterForm 对象收集用户的请求参数值，并可以运用它与 Spring 绑定标签合作，当相关的验证失败时在网页上显示错误消息，RegisterForm 的设计如下所示：

SpringBookmark　　RegisterForm.java

```
package onlyfun.caterpillar.web;

public class RegisterForm {
    private String email;
    private String username;
    private String passwd;
    private String passwd2;

    public void setEmail(String email) {
        this.email = email;
    }

    public void setUsername(String username) {
        this.username = username;
    }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }

    public void setPasswd2(String passwd2) {
        this.passwd2 = passwd2;
    }

    public String getEmail() {
        return email;
    }

    public String getUsername() {
        return username;
    }

    public String getPasswd() {
        return passwd;
    }

    public String getPasswd2() {
        return passwd2;
    }
}
```

用户在填写了注册表单并送出之后，需要作一些验证检查动作，例如检查用户的邮件格式是否正确，密码长度是否在 6 个字符至 16 个字符之间，两次输入的密码是否相同等，这可以使用一个 RegisterValidator 对象来完成相关的验证检查动作，这个对象实现了 Spring 的 Validator 接口（参考 7.3.1 节），编写方式如下所示：

SpringBookmark　　RegisterValidator.java

```
package onlyfun.caterpillar.web;

import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

public class RegisterValidator implements Validator {
    public boolean supports(Class clazz) {
        return clazz.equals(RegisterForm.class);
    }

    public void validate(Object obj, Errors errors) {
        RegisterForm form = (RegisterForm) obj;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        if(!validateEmail(form.getEmail())) {
            errors.rejectValue("email",
                    "invalid_email", null, "邮件格式不正确");
        }

        if(form.getPasswd().length() < 6 ||
          form.getPasswd().length() > 16) {
            errors.rejectValue("passwd",
                "less-or-over", null, "密码长度必须是 6 到 16 字符");
        }

        if(!form.getPasswd().equals(form.getPasswd2())) {
            errors.rejectValue(
              "passwd", "not-equal", null, "两次输入的密码不符");
        }
    }

    private boolean validateEmail(String email) {
        return email.matches("^[_a-z0-9-]+([.]" +
                "[_a-z0-9-]+)*@[a-z0-9-]+([.][a-z0-9-]+)*$");
    }
}
```

RegisterController 类将继承 Spring 的 SimpleFormController 类（参考 7.2.7 节），这个 Validator 对象可以直接使用 Spring 的依赖注入来设置至 Controller，来看一下 RegisterController 类的编写：

SpringBookmark　　RegisterController.java

```
package onlyfun.caterpillar.web;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import onlyfun.caterpillar.model.IUserDAO;
import onlyfun.caterpillar.model.User;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.
                            mvc.SimpleFormController;

public class RegisterController extends SimpleFormController {
    private IUserDAO userDAO;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public RegisterController() {
        setCommandClass(RegisterForm.class);
    }

    protected ModelAndView onSubmit(HttpServletRequest request,
                    HttpServletResponse response,
                    Object command,
                    BindException exception) throws Exception {
        RegisterForm form = (RegisterForm) command;

        User user = new User();

        user.setEmail(form.getEmail());
        user.setUsername(form.getUsername());
        user.setPasswd(form.getPasswd());

        this.getUserDAO().insert(user);
        request.getSession().setAttribute(
               "valid_user", form.getUsername());

        return new ModelAndView(this.getSuccessView());
    }

    public void setUserDAO(IUserDAO userDAO) {
        this.userDAO = userDAO;
    }

    public IUserDAO getUserDAO() {
        return userDAO;
    }
}
```

在 RegisterController 中，将从 RegisterForm 中撷取相关信息，并设置至一个 User 的实例中，接下来取得 IUserDAO 对象，以将 User 对象中的消息储存至数据库的 user 表格，注意到 RegisterForm 是属于 Web 层所设计的 API，虽然它当中就带有用户的注册消息，但不建议直接将 Web 层所设计的 API 侵入至业务层之中，而改用一个与 Web 层无关的 User 对象来封装用户注册消息。

IUserDAO 的实例将使用 Spring 的依赖注入功能设置，注册完成之后，在 Session 对象的属性中设置 "valid_user"，用户是否登录即是根据这个属性来进行判断的。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

10.4.2　LoginController

在用户进行登录的时候，可以使用一个 LoginForm 对象来收集用户的请求参数值，并可以运用它与 Spring 绑定标签合作，在登录失败时于网页上显示错误消息，LoginForm 的设计如下所示：

SpringBookmark　　LoginForm.java

```
package onlyfun.caterpillar.web;

public class LoginForm {
    private String username;
    private String passwd;

    public void setUsername(String username) {
        this.username = username;
    }

    public void setPasswd(String passwd) {
        this.passwd = passwd;
    }

    public String getUsername() {
        return username;
    }

    public String getPasswd() {
        return passwd;
    }
}
```

接着设计 LoginController 类，在处理登录时，会检查用户名称与密码是否符合，如果符合的话，在 Session 对象的属性中设置 "valid_user"，用户是否登录即是根据这个属性来进行判断的，如果不符合则设置相关错误消息并送回至登录页面，LoginController 类的编写如下所示：

SpringBookmark　　LoginController.java

```
package onlyfun.caterpillar.web;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import onlyfun.caterpillar.model.IUserDAO;
import onlyfun.caterpillar.model.User;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.
                        mvc.SimpleFormController;

public class LoginController extends SimpleFormController {
   private IUserDAO userDAO;

   public LoginController() {
      this.setCommandClass(LoginForm.class);
   }

   protected ModelAndView onSubmit(HttpServletRequest request,
                     HttpServletResponse response,
                     Object command,
                     BindException exception)
                                    throws Exception {
      LoginForm form = (LoginForm) command;

      User user = userDAO.findByName(form.getUsername());

      if(user == null) {
         exception.rejectValue("username",
                "no_such_user", "查无此人");
         return new ModelAndView(
                this.getFormView(), exception.getModel());
      }
      else if(user != null &&
            !user.getPasswd().equals(form.getPasswd())) {
         exception.rejectValue("passwd",
                "wrong_passwd", "密码错误");
         return new ModelAndView(
                this.getFormView(), exception.getModel());
      }
      else {
         request.getSession().setAttribute(
                "valid_user", user.getUsername());
         return new ModelAndView(this.getSuccessView());
      }
   }

   public void setUserDAO(IUserDAO userDAO) {
      this.userDAO = userDAO;
   }
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    public IUserDAO getUserDAO() {
        return userDAO;
    }
}
```

10.4.3 MemberController

会员登录之后，接下来可以进行个人书签的显示、加入或删除书签、更改密码、登出等动作，这些动作都是属于会员才会拥有的功能，因此为了集合管理这些操作，可以继承 Spring 的 MultiActionController 类来编写这些操作的处理（参考 7.2.2 节），而不用为每个操作编写一个 Controller 类，来看一下 MemberController 的编写：

SpringBookmark　　MemberController.java

```
package onlyfun.caterpillar.web;

import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import onlyfun.caterpillar.model.IBookmarkDAO;
import onlyfun.caterpillar.model.IUserDAO;
import onlyfun.caterpillar.model.User;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.
               multiaction.MultiActionController;
import org.springframework.web.servlet.view.RedirectView;

public class MemberController extends MultiActionController {
    private IBookmarkDAO bookmarkDAO;
    private IUserDAO userDAO;
    private String memberPage;
    private String addBmPage;
    private String passwdPage;
    private String redirectPage;

    // 显示个人书签
    public ModelAndView displayUrl(HttpServletRequest req,
                                   HttpServletResponse res) {
        String username =
          (String) req.getSession().getAttribute("valid_user");

        if(username != null) {
            // 根据用户名称查找书签
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            List urls =
              getBookmarkDAO().findUserUrl(username);
            return new ModelAndView(
                  this.getMemberPage(), "urls", urls);
        }
        else {
            return new ModelAndView(
                  new RedirectView(getRedirectPage()));
        }
    }

    // 加入新的书签
    public ModelAndView addUrl(HttpServletRequest req,
                        HttpServletResponse res) {
        String username =
            (String) req.getSession().getAttribute("valid_user");

        if(username != null) {
            if(req.getParameter("new_url") != null) {
                String url = (String) req.getParameter("new_url");
                // 加入新的书签网址
                getBookmarkDAO().addUserUrl(username, url);
                return displayUrl(req, res);
            }
            else {
                return new ModelAndView(this.getAddBmPage());
            }
        }
        else {
            return new ModelAndView(
                  new RedirectView(getRedirectPage()));
        }
    }

    // 删除所选择的书签
    public ModelAndView deleteUrl(HttpServletRequest req,
                          HttpServletResponse res) {
        String username =
          (String) req.getSession().getAttribute("valid_user");

        if(username != null) {
            if(req.getParameterValues("del_me") != null) {
                String[] deleted_urls =
                    (String[]) req.getParameterValues("del_me");
                // 删除所指定的书签
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
            getBookmarkDAO().deleteUserUrls(
                        username, deleted_urls);
            return displayUrl(req, res);
        }
        else {
            return new ModelAndView(this.getAddBmPage());
        }
    }
    else {
        return new ModelAndView(
               new RedirectView(getRedirectPage()));
    }
}

// 改变密码
public ModelAndView changePasswd(HttpServletRequest req,
                          HttpServletResponse res) {
    String username =
        (String) req.getSession().getAttribute("valid_user");

    if(username != null) {
        if(req.getParameter("new_passwd") != null) {
            String passwd = req.getParameter("new_passwd");
            User user = getUserDAO().findByName(username);
            user.setPasswd(passwd);
            getUserDAO().update(user);
            return displayUrl(req, res);
        }
        else {
            return new ModelAndView(this.getPasswdPage());
        }
    }
    else {
        return new ModelAndView(
               new RedirectView(getRedirectPage()));
    }
}

// 登出
public ModelAndView logout(HttpServletRequest req,
                     HttpServletResponse res) {
    req.getSession().invalidate();
    return new ModelAndView(
           new RedirectView(getRedirectPage()));
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
}

public void setBookmarkDAO(IBookmarkDAO bookmarkDAO) {
    this.bookmarkDAO = bookmarkDAO;
}

public IBookmarkDAO getBookmarkDAO() {
    return bookmarkDAO;
}

public void setUserDAO(IUserDAO userDAO) {
    this.userDAO = userDAO;
}

public IUserDAO getUserDAO() {
    return userDAO;
}

public String getMemberPage() {
    return memberPage;
}

public void setMemberPage(String memberPage) {
    this.memberPage = memberPage;
}

public String getAddBmPage() {
    return addBmPage;
}

public void setAddBmPage(String addBmPage) {
    this.addBmPage = addBmPage;
}

public String getPasswdPage() {
    return passwdPage;
}

public void setPasswdPage(String passwdPage) {
    this.passwdPage = passwdPage;
}

public String getRedirectPage() {
    return redirectPage;
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    }

    public void setRedirectPage(String redirectPage) {
        this.redirectPage = redirectPage;
    }
}
```

进行相关操作时，需要 IUserDAO 及 IBookmarkDAO 的实例，这可以使用 Spring 的依赖注入来进行设置。

10.4.4 ForgotController

当用户忘记密码而无法登录时，可以输入注册时所使用的邮件地址，系统会发一封信至指定的信箱，以上这些动作是在 ForgotController 类中定义的，来看一下它是如何编写的：

SpringBookmark　　ForgotController.java

```
package onlyfun.caterpillar.web;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import onlyfun.caterpillar.model.IUserDAO;
import onlyfun.caterpillar.model.SimpleMail;
import onlyfun.caterpillar.model.User;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.
                        mvc.AbstractController;

public class ForgotController extends AbstractController {
    private IUserDAO userDAO;
    private SimpleMail simpleMail;
    private String loginPage;
    private String forgotPage;

    public ModelAndView handleRequestInternal(
            HttpServletRequest request,
            HttpServletResponse response) throws Exception {
        String email = request.getParameter("email");
        if(email != null && !"".equals(email)) {
            // 根据邮件地址查询用户的数据
            User user = getUserDAO().findByEmail(email);
            // 取得 SimpleMail 实例并设置相关数据以发送邮件
            getSimpleMail().sendPasswdMail(
                    user.getEmail(),
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
                user.getUsername(),
                user.getPasswd());
            return new ModelAndView(this.getLoginPage());
        }
        else {
            return new ModelAndView(this.getForgotPage());
        }
    }

    public void setUserDAO(IUserDAO userDAO) {
        this.userDAO = userDAO;
    }

    public IUserDAO getUserDAO() {
        return userDAO;
    }

    public void setSimpleMail(SimpleMail simpleMail) {
        this.simpleMail = simpleMail;
    }

    public SimpleMail getSimpleMail() {
        return simpleMail;
    }

    public void setLoginPage(String loginPage) {
        this.loginPage = loginPage;
    }

    public String getLoginPage() {
        return loginPage;
    }

    public void setForgotPage(String forgotPage) {
        this.forgotPage = forgotPage;
    }

    public String getForgotPage() {
        return forgotPage;
    }
}
```

同样的，相关的依赖对象也是使用 Spring 的依赖注入来进行设置。

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

10.5　配置设置

Spring 在线书签所需要的页面与类等组件已编写完成，现在要将这一切串接在一起，让程序中的各个组件可以彼此合作，方法就是编写一些 XML 文件进行配置设置，让 Spring 可以使用其 Web MVC 框架与容器功能，将整个程序运行起来。

10.5.1　设置 web.xml

在 *web.xml* 中要设置 Spring 的 `DispatcherServlet`（参考 7.1.2 节），并告知 Spring 的 XML 设置文件之名称与所在位置，来看一下 *web.xml* 中是如何编写的：

SpringBookmark　　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   → http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>

    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/web-config.xml,
                    → /WEB-INF/model-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>*.do</url-pattern>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
    </servlet-mapping>
</web-app>
```

为了避免与 Spring 相关的设置文件内容过长，因此将 Web 层的 Controller 设置、View 设置等在 *web-config.xml* 中进行设置，而将相关的 Model 对象设置于 *model-config.xml* 之中。

10.5.2 设置 web-config.xml

在 *web-config.xml* 中主要是设置 HandlerMapping、ViewResolver、Controller、View 等内容（请参考 7.1 节相关的内容），直接来看看如何编写：

SpringBookmark　　web-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <bean id="urlMapping"
         class="org.springframework.web.servlet.
                → handler.SimpleUrlHandlerMapping">
       <property name="mappings">
          <props>
             <prop key="/login.do">loginController</prop>
             <prop key="/register.do">registerController</prop>
             <prop key="/member.do">memberController</prop>
             <prop key="/forgot.do">forgotController</prop>
          </props>
       </property>
    </bean>

    <bean id="viewResolver"
         class="org.springframework.web.servlet.
                → view.InternalResourceViewResolver">
       <property name="viewClass"
          value="org.springframework.web.servlet.view.JstlView"/>
       <property name="prefix" value="/WEB-INF/jsp/"/>
       <property name="suffix" value=".jsp"/>
    </bean>

    <bean id="loginController"
         class="onlyfun.caterpillar.web.LoginController">
       <property name="userDAO" ref="userDAO"/>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        <property name="successView" value="login_success"/>
        <property name="formView" value="login"/>
    </bean>

    <bean id="registerValidator"
        class="onlyfun.caterpillar.web.RegisterValidator"/>

    <bean id="registerController"
        class="onlyfun.caterpillar.web.RegisterController">
        <property name="validator" ref="registerValidator"/>
        <property name="userDAO" ref="userDAO"/>
        <property name="successView" value="register_success"/>
        <property name="formView" value="register"/>
    </bean>

    <bean id="paraMethodResolver"
        class="org.springframework.web.servlet.mvc.
                ➥ multiaction.ParameterMethodNameResolver">
        <property name="paramName" value="action"/>
        <property name="defaultMethodName" value="displayUrl"/>
    </bean>

    <bean id="memberController"
        class="onlyfun.caterpillar.web.MemberController">
        <property name="methodNameResolver"
            ref="paraMethodResolver"/>
        <property name="bookmarkDAO" ref="bookmarkDAO"/>
        <property name="userDAO" ref="userDAO"/>
        <property name="memberPage" value="member"/>
        <property name="addBmPage" value="addbm"/>
        <property name="passwdPage" value="change_passwd"/>
        <property name="redirectPage"
            value="/SpringBookmark/login.do"/>
    </bean>

    <bean id="forgotController"
        class="onlyfun.caterpillar.web.ForgotController">
        <property name="userDAO" ref="userDAO"/>
        <property name="simpleMail" ref="simpleMail"/>
        <property name="loginPage" value="login"/>
        <property name="forgotPage" value="forgot"/>
    </bean>
</beans>
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

10.5.3 设置 model-config.xml

在 *model-config.xml* 中，则是编写数据库的基本消息、Hibernate 相关的配置，以及 UserDAO、BookmarkDAO 的定义（参考 6.2 节的内容），另外有关 SimpleMail 的定义，也在这个文件中编写：

SpringBookmark　　model-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <!-- 要设置您的数据库相关消息 -->
    <bean id="dataSource"
         class="org.springframework.jdbc.
                → datasource.DriverManagerDataSource">
        <property name="driverClassName"
                value="com.mysql.jdbc.Driver"/>
        <property name="url"
            value="jdbc:mysql://localhost:3306/bookmarks"/>
        <property name="username" value="your_db_username"/>
        <property name="password" value="your_db_password"/>
    </bean>

    <bean id="sessionFactory"
         class="org.springframework.orm.
                → hibernate3.LocalSessionFactoryBean"
         destroy-method="destroy">
        <property name="dataSource" ref="dataSource"/>
        <property name="mappingResources">
            <list>
             <value>onlyfun/caterpillar/model/User.hbm.xml</value>
             <value>onlyfun/caterpillar/model/Bookmark.hbm.xml</value>
             </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">
                    org.hibernate.dialect.MySQLDialect
                </prop>
            </props>
        </property>
    </bean>

    <bean id="hibernateTemplate"
```


Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。

```
        class="org.springframework.orm.
                → hibernate3.HibernateTemplate">
      <property name="sessionFactory" ref="sessionFactory"/>
    </bean>

    <bean id="userDAO"
        class="onlyfun.caterpillar.model.UserDAO">
      <property name="hibernateTemplate" ref="hibernateTemplate"/>
    </bean>

    <bean id="bookmarkDAO"
        class="onlyfun.caterpillar.model.BookmarkDAO">
      <property name="hibernateTemplate" ref="hibernateTemplate"/>
    </bean>

    <!-- 要设置您的 Smtp Server 与寄件人 -->
    <bean id="simpleMail"
        class="onlyfun.caterpillar.model.SimpleMail">
      <property name="smtpHost" value="your_smtp_server"/>
      <property name="from" value="your_admin_address"/>
    </bean>
</beans>
```

到这里为止，Spring 在线书签所必须的文件都已编写完成，最后别忘了要设置相关的 *.jar* 文件，然后就可以开始运行您的程序了，可以在光盘中找到 SpringBookmark 项目，并修改当中的一些个人化配置，例如数据库用户名称、密码等，以及 Mail 的相关设置，然后使用 WTP 运行程序，就可以看看 Spring 在线书签运行时的功能。

10.6　接下来的主题

本书到这里要告一段落了，最后要补充一点的是，在 Spring 下载文件中有一个 *samples* 目录，当中有一些 Spring 的范例程序可以参考，建议您可以看一下 *jpetstore* 目录下的范例程序，这也是一个观摩如何使用 Spring 相当好的范例。



图 10.10　试着用 Spring 写个程序吧

Java学习群：72030155
好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间；可以联系群主获取更多大型企业内部技术教程。